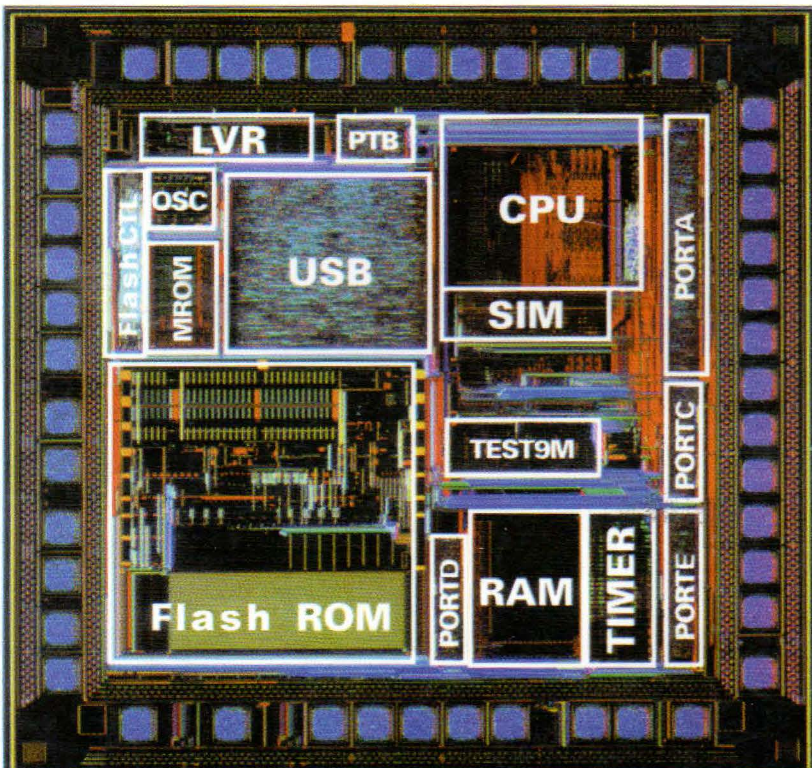
刘慧银 程建平 龚光华 王云飞 许庆丰 编著

Motorola 微控制器 MC68HC08 原理及其嵌入式应用



清华大学出版社
http://www.tup.tsinghua.edu.cn

果粉工作室 Retro Workshop

Motorola 微控制器 MC68HC08 原理及其嵌入式应用

刘慧银　程建平　龚光华　王云飞　许庆丰　编著

清华大学出版社

(京)新登字 158 号

内 容 提 要

本书是清华大学“微控制器原理及其嵌入式应用”课程的教材，它以 MC68HC908GP32 为范例介绍了 Motorola 高档 8 位微控制器的 CPU、存储器以及 I/O 接口模块的结构和功能，汇编语言和 C 语言编程，开发方法和开发工具等内容，尤其对 Flash 编程、锁相环原理及编程、监控 ROM 及嵌入式操作系统 μC/OS Ⅱ做了详细说明。此外，本书还讨论了嵌入式微控制器系统抗干扰和电磁兼容性问题，并提供了印刷线路板的设计经验。书中大量的程序范例可直接引用。

本书是为清华大学等十余所大学的本科生和研究生编写的教材，也可供用微控制器开发产品的工程技术人员和业余爱好者参考。

版权所有，翻印必究。
本书封面贴有清华大学出版社激光防伪标签，无标签者不得销售。

书　名：Motorola 微控制器 MC68HC08 原理及其嵌入式应用
作　者：刘慧银　程建平　龚光华　王云飞　许庆丰　编著
出版者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
　　　　http://www.tup.tsinghua.edu.cn
印刷者：北京国马印刷厂
发行者：新华书店总店北京发行所
开　本：787×1092 1/16　印张：20　字数：460 千字
版　次：2001 年 8 月第 1 版　2001 年 8 月第 1 次印刷
书　号：ISBN 7-302-04678-6/TP・2779
印　数：0001～5000
定　价：26.00 元

前　言

Motorola 微控制器 MC68HC05 自问世以来，被广泛应用于工业、农业、商业、军工、医疗、通信、化工、材料等几乎所有国民经济领域，特别是在家电、耐用消费品等人民生活领域占据了广阔的市场。它有三十几个系列、二百多个品种。然而 1999 年中期，Motorola公司正式在中国推出了 MC68HC08，以逐步实现 MC68HC05 各系列产品的升级换代。

Motorola 公司是世界上最大的微控制器类集成电路供应商。随着 Motorola 公司在中国投资的增加以及该公司的本土化政策，其嵌入式应用产品得到越来越多认同。除移动通信外，微控制器类集成电路的设计、生产及应用开发将逐步实现本土化。Motorola 公司的技术设计中心、芯片生产企业也相继建立在中国。虽然目前 MC68HC08 的产品系列还不如 MC68HC05 多，但再过几年，MC68HC05 的几十个系列终将升级到 MC68HC08。

从 MC68HC05 到 MC68HC08 的产品升级与优化是多方面的，然而我们认为最重要的、革命性的优化在以下 3 个方面。

(1) 锁相环技术的应用使外部时钟频率降到 32kHz，而内部时钟可达 32MHz。这一改进大大降低了应用系统的噪声，提高了应用系统的抗电磁干扰能力。

(2) Flash 技术的应用使应用程序的在线编程成为可能，可以说是过去 OTP(一次性可编程)型和掩膜型产品的一场革命，使得应用类产品的开发不再需要昂贵的仿真器。

(3) 与 MC68HC05 向上兼容，增加了 78 条指令，并大大优化了 MC68HC08 CPU 指令集，特别是将 MC68HC05 固定的 8 位栈指针优化为 16 位浮动栈指针，使得高级语言，例如 C 语言的应用成为可能。随着推广使用高级语言，多年来我国微控制器应用开发人员所熟知的汇编语言将逐步被忘记。

以上三大革命性优化，也正是用 MC68HC08 微控制器开发嵌入式应用产品的三大难点。而这三大难点正是本书的三个重点内容。

本书从介绍微控制器与微处理器的特点、区别入手，重点介绍了 CPU08 的结构、指令系统、C 语言及汇编语言编程，并以 MC68HC908GP32(其中央处理器为 CPU08)为例系统介绍了高档 8 位微控制器的基本组成模块，包括中断与复位、存储器组织与空间分配、定时器结构与双定时器的使用、并行 I/O 与同步异步串行 I/O 等；详细介绍了独具特色的 Flash 编程原理与过程，锁相环电路的原理、参数计算与编程方法，微控制器自带的监控 ROM，以及在低功耗模式下微控制器各模块的工作状态。本书还通过 2 个例子说明嵌入式微控制器系统的设计思路和调试方法，并介绍了在研制微控制器系统尤其是产品化过程中需要重点考虑的电磁兼容性问题，以及印刷电路板设计与布线、元件的选择等实用经验。此外，本书在介绍嵌入式实时系统在 MC68HC08 上的实现的同时，还全面介绍了微控制器的各种开发方法，详细介绍了清华 Motorola 单片机应用开发研究中心自行研

制的 MC68HC908GP32 IDK 在线编程开发系统的工作原理、硬件模块，以及程序的编辑、编译和调试方法，并提供了大量应用范例。

本书共计 15 章，其中刘慧银编写了第 2 章，程建平编写了第 4 章，龚光华编写了第 6，11 章和第 14.1 节，王云飞编写了第 5，15 章和第 8.3，8.4 节，许庆丰编写了第 7，10 章和第 8.1，14.2 节，薛涛编写了第 3 章，郑杨斌编写了第 8.2 节，邵贝贝编写了第 1，12，13 章。全书由刘慧银统稿，吴凤茹负责录入、校对、排版工作。

本书是为清华大学等十余所大学的本科生和研究生编写的教材，也可供从事 Motorola微控制器应用与开发的工程技术人员参考。

本书在编写过程中，Motorola 半导体亚太区总部的刘显礼先生、黄耀君先生、冯启明先生、汤儒蓉女士等向我们提供了详尽的英文技术资料，多次为我们解答开发过程中的疑难问题。特别是该公司的金功九博士，在 MC68HC908GP32 IDK 在线开发系统的设计与制造方面为我们提供了强有力的支持。特在此表示深切谢意。

由于带有 Flash 存储器的 MC68HC08 是一个推出不久的新型微控制器，我们对它的理解存在局限性，特别是一些好心的朋友不断催促这本书早日出版，书中的缺欠和错误在所难免，请广大读者批评指正。

清华 Motorola 单片机应用开发研究中心

2001 年 4 月

目　　录

第 1 章　微控制器概论 …… 1

1.1　什么是微控制器 …… 1
1.2　微控制器与微处理器——微计算机技术的两大分支 …… 3
1.3　微控制器与专用集成电路 …… 4
1.4　微控制器的历史与发展 …… 5
1.5　微控制器的开发手段 …… 6
1.6　Motorola 微控制器命名法 …… 7

第 2 章　新一代 8 位微控制器 MC68HC08 …… 11

2.1　MC68HC08 系列微控制器概述 …… 11
2.1.1　MC68HC08 各系列简介 …… 11
2.1.2　MC68HC08 系列微控制器的主要特点 …… 12
2.1.3　MC68HC08 中央处理器 CPU08 …… 12
2.2　MC68HC908GP32 概述 …… 18
2.2.1　MC68HC908GP32 的特性 …… 18
2.2.2　MC68HC908GP32 的功能结构 …… 19
2.2.3　MC68HC908GP32 的存储器组织与空间分配 …… 25
2.2.4　MC68HC908GP32 的系统设置 …… 27
2.2.5　MC68HC908GP32 的管脚与封装 …… 28

第 3 章　中断与复位 …… 30

3.1　中断 …… 30
3.1.1　中断的效果 …… 30
3.1.2　中断源 …… 30
3.1.3　中断状态寄存器 …… 34
3.2　复位 …… 35
3.2.1　复位的效果 …… 36
3.2.2　外部复位 …… 36
3.2.3　内部复位 …… 36
3.2.4　SIM 复位状态寄存器 …… 37
3.3　CONFIG 寄存器 …… 38
3.4　系统集成模块简介 …… 39

第 4 章　MC68HC08 指令系统及汇编语言程序设计 …… 41

4.1　指令分类 …… 41
4.2　寻址方式 …… 41
4.2.1　隐含寻址方式 …… 43
4.2.2　立即寻址方式 …… 43
4.2.3　直接寻址方式 …… 43
4.2.4　扩展寻址方式 …… 43
4.2.5　变址寻址方式 …… 44
4.2.6　相对寻址方式 …… 45
4.2.7　存储器到存储器的寻址方式 …… 45
4.3　汇编语言程序设计 …… 46
4.3.1　机器语言和汇编语言 …… 46
4.3.2　汇编语言源程序的格式 …… 47
4.3.3　汇编伪指令 …… 49
4.3.4　汇编语言程序的编译 …… 51
4.3.5　S 记录 …… 51
4.3.6　汇编语言程序设计和举例 …… 53

第 5 章　闪速存储器 …… 60

5.1　闪速存储器概述 …… 60
5.2　闪速存储器的编程操作 …… 61
5.2.1　闪速存储器编程操作概述 …… 61
5.2.2　闪速存储器的编程寄存器 …… 61
5.2.3　闪速存储器的编程步骤 …… 63
5.2.4　闪速存储器的编程范例 …… 65
5.3　闪速存储器的编程模式 …… 70
5.3.1　闪速存储器编程模式概述 …… 70
5.3.2　用户模式下的闪速存储器编程方法 …… 71

第 6 章　定时器 …… 72

6.1　定时器的结构及基本功能 …… 72
6.1.1　定时器的寄存器 …… 72
6.1.2　定时器溢出 …… 74
6.1.3　定时器预置计数溢出 …… 74
6.1.4　定时器的状态和控制寄存器 …… 75
6.2　输入捕捉 …… 77
6.2.1　输入捕捉的概念 …… 77

6.2.2 输入捕捉操作…… 78
6.2.3 输入捕捉的控制寄存器…… 78
6.2.4 输入捕捉编程实例…… 80
6.3 输出比较…… 81
6.3.1 输出比较的概念…… 81
6.3.2 输出比较操作…… 81
6.3.3 输出比较的控制寄存器…… 82
6.3.4 不带缓冲的输出比较…… 83
6.3.5 带缓冲的输出比较…… 84
6.4 脉宽调制输出…… 85
6.4.1 脉宽调制的概念…… 85
6.4.2 不带缓冲的脉宽调制波…… 86
6.4.3 带缓冲的脉宽调制波…… 88

第7章 时钟发生模块 …… 90

7.1 通用锁相环频率合成器简介…… 90
7.1.1 通用锁相环频率合成器的系统结构…… 90
7.1.2 通用锁相环频率合成器的主要技术指标…… 91
7.2 MC68HC08 时钟发生模块 …… 92
7.2.1 MC68HC08 时钟发生模块概述 …… 92
7.2.2 MC68HC08 时钟发生模块的内部组成 …… 93
7.2.3 MC68HC08 时钟发生模块的外部引脚 …… 95
7.2.4 MC68HC08 时钟发生模块的晶振电路 …… 96
7.2.5 MC68HC08 时钟发生模块的锁相环频率合成器 …… 97
7.2.6 MC68HC08 时钟发生模块的时钟选择电路 …… 98
7.3 MC68HC908GP32 时钟发生模块 …… 99
7.3.1 MC68HC908GP32 时钟发生模块的寄存器 …… 99
7.3.2 MC68HC908GP32 时钟发生模块中锁相环电路参数计算 …… 104
7.3.3 MC68HC908GP32 时钟发生模块编程举例 …… 107
7.3.4 MC68HC908GP32 时钟发生模块在 WAIT、STOP 指令中的状态 …… 108

第8章 输入输出 I/O …… 110

8.1 并行 I/O …… 110
8.1.1 并行口 …… 110
8.1.2 并行口应用举例 …… 112
8.2 异步串行通信接口 SCI …… 113
8.2.1 RS-232C 标准 …… 114
8.2.2 标准不归零数据格式 …… 115

8.2.3 SCI 功能 …………………………………………………… 116
8.2.4 SCI 发送器 …………………………………………………… 117
8.2.5 SCI 接收器 …………………………………………………… 117
8.2.6 SCI 寄存器 …………………………………………………… 118
8.2.7 SCI 应用 …………………………………………………… 123
8.2.8 软件串行口 …………………………………………………… 126
8.3 同步串行外设接口 SPI …………………………………………………… 130
8.3.1 SPI 的数据流动 …………………………………………………… 131
8.3.2 SPI 功能 …………………………………………………… 133
8.3.3 SPI 引脚 …………………………………………………… 133
8.3.4 SPI 寄存器 …………………………………………………… 135
8.3.5 SPI 应用举例 …………………………………………………… 138
8.4 A/D 转换器 …………………………………………………… 141
8.4.1 A/D 状态与控制寄存器 …………………………………………………… 141
8.4.2 A/D 数据寄存器 …………………………………………………… 142
8.4.3 A/D 时钟寄存器 …………………………………………………… 143

第 9 章 用 C 语言开发 MC68HC08 微控制器系统应用程序 … 144

9.1 C 语言是开发微控制器应用系统软件的有力工具 …………………………………………………… 144
9.1.1 使用 C 语言编写应用程序 …………………………………………………… 144
9.1.2 使用汇编语言编写与硬件相关的程序 …………………………………………………… 145
9.2 交叉编译和 C 语言程序运行环境的建立 …………………………………………………… 145
9.3 应用程序的构成与模块化程序结构 …………………………………………………… 147
9.4 全程变量与局部变量 …………………………………………………… 147
9.5 函数的结构与函数间参数的传递 …………………………………………………… 148
9.6 程序模块的框架与组织 …………………………………………………… 150
9.7 宏调用 …………………………………………………… 151

第 10 章 低功耗模式 …………………………………………………… 152

10.1 WAIT 和 STOP 指令 …………………………………………………… 152
10.1.1 STOP 指令 …………………………………………………… 152
10.1.2 WAIT 指令 …………………………………………………… 152
10.2 唤醒微控制器的事件 …………………………………………………… 154
10.3 在 WAIT 和 STOP 模式下微控制器中各模块的工作状态 …………………………………………………… 155
10.3.1 A/D 转换模块 …………………………………………………… 155
10.3.2 断点模块(BRK) …………………………………………………… 155
10.3.3 中央处理器(CPU) …………………………………………………… 155
10.3.4 时钟发生模块(CGM) …………………………………………………… 155

10.3.5 看门狗模块(COP) …… 156
10.3.6 外中断模块(IRQ) …… 156
10.3.7 键盘中断模块(KBI) …… 156
10.3.8 低电压禁止模块(LVI) …… 156
10.3.9 异步串行通信接口(SCI) …… 156
10.3.10 同步串行通信接口(SPI) …… 156
10.4 低功耗应用举例 …… 157
10.4.1 智能二线制仪表简介 …… 157
10.4.2 智能二线制仪表的硬件电路 …… 158
10.4.3 智能二线制仪表的低功耗措施 …… 159
10.4.4 智能二线制仪表的软件 …… 160
10.4.5 智能二线制仪表系统功耗的估算 …… 161

第 11 章 监控 ROM 程序 …… 163

11.1 监控 ROM 程序的功能 …… 163
11.2 监控 ROM 程序详解 …… 164

第 12 章 电磁兼容性 …… 174

12.1 电磁兼容的基本概念 …… 174
12.2 电磁兼容组织与标准 …… 174
12.3 微控制器系统的电磁兼容问题 …… 176
12.4 噪声的来源与传输 …… 177
12.5 信号线间的交叉干扰 …… 179
12.6 来自电源的噪声 …… 180
12.7 印刷线路板与元器件的高频特性 …… 180
12.8 元件的布置 …… 181
12.9 印刷线路板接地线的处理 …… 182
12.10 抑制噪声的器件 …… 183
12.11 控制噪声的经验 …… 184
12.11.1 控制噪声源 …… 184
12.11.2 减小噪声的耦合 …… 185
12.11.3 减小噪声接收 …… 186

第 13 章 嵌入式实时系统在 MC68HC08 上的实现 …… 187

13.1 什么是嵌入式实时操作系统 …… 187
13.2 3 种嵌入式实时系统 …… 187
13.2.1 前后台系统 …… 188
13.2.2 使用不可剥夺型内核的多任务实时系统 …… 188

13.2.3 使用可剥夺型内核的多任务实时系统 …… 189
13.3 使用嵌入式实时操作系统的优点和缺点 …… 189
13.3.1 使用嵌入式实时操作系统的优点 …… 189
13.3.2 使用嵌入式实时操作系统的缺点 …… 189
13.4 在 MC68HC08 CPU 上实现实时内核的运行 …… 190
13.5 在 MC68HC08 CPU 上运行一个基于优先级的不可剥夺型实时内核 …… 190
13.6 在 MC68HC08 CPU 上运行一个基于时间片的不可剥夺型实时内核 …… 203
13.7 在 MC68HC08 CPU 上运行可剥夺型实时内核 μC/OS-Ⅱ …… 208
13.7.1 重新定义内核的大小和功能 …… 209
13.7.2 修改内核代码的汇编部分 …… 211
13.7.3 编写用户应用程序 …… 216

第 14 章 设计应用举例 …… 219

14.1 智能电话门铃 …… 219
14.1.1 硬件设计 …… 219
14.1.2 软件流程 …… 221
14.2 MC68HC908GP32 在智能 UPS 电源中的应用 …… 222
14.2.1 硬件结构 …… 223
14.2.2 系统软件流程 …… 225

第 15 章 MC68HC908GP32 IDK 在线编程开发系统 …… 227

15.1 MC68HC908GP32 IDK 概述 …… 228
15.1.1 MC68HC908GP32 IDK 的特点 …… 228
15.1.2 MC68HC908GP32 IDK 套件 …… 228
15.1.3 MC68HC908GP32 IDK 对 PC 机的要求 …… 228
15.2 MC68HC908GP32 IDK 的硬件及与 PC 机的连接 …… 228
15.2.1 MC68HC908GP32 IDK 的硬件模块 …… 229
15.2.2 MC68HC908GP32 IDK 与 PC 机的连接 …… 238
15.3 MC68HC908GP32 IDK 使用指南 …… 239
15.3.1 监控命令列表 …… 240
15.3.2 监控命令描述 …… 241
15.3.3 MC68HC908GP32 IDK 使用注意事项 …… 244
15.4 MC68HC908GP32 IDK 的软件编程 …… 245
15.4.1 程序代码和变量地址分配 …… 245
15.4.2 程序初始化 …… 245
15.4.3 MC68HC908GP32 IDK 中的用户程序与目标系统中用户程序的关系 …… 247
15.5 用户程序的编辑、编译及调试 …… 248

15.5.1 编辑源代码 …… 248
15.5.2 编译源程序 …… 249
15.5.3 将编译好的 S 记录格式文件调整为适合下载的格式 …… 249
15.5.4 与核心子板建立通信 …… 250
15.5.5 下载程序 …… 250
15.5.6 设定程序运行起点并开始调试 …… 250
15.6 终端仿真程序 PROCOMM 的使用说明 …… 250
15.6.1 PROCOMM 简介 …… 250
15.6.2 PROCOMM 设置 …… 252
15.6.3 PROCOMM 通信 …… 252
15.7 MC68HC908GP32 IDK 集成开发环境使用说明 …… 254
15.7.1 8 位 Motorola 微控制器汇编程序编译环境 …… 254
15.7.2 终端仿真程序——超级终端 …… 255

附录 …… 259

1. MC68HC908GP32 引脚图 …… 259
2. 存储器分配图 …… 262
3. 寄存器表 …… 263
4. 中断矢量表 …… 271
5. MC68HC08 指令集操作表 …… 272
6. MC68HC08 指令系统详细说明 …… 274
7. 参数极值表 …… 302
8. 直流电气特性 …… 303
9. 控制时序表 …… 306

参考文献 …… 307

第1章　微控制器概论

1.1　什么是微控制器

微控制器在国内有个更通俗的名字——单片机。

单片机一词源于英文的 single chip microcomputer，意为单片微计算机。我们知道，计算机是由三大部分组成的：中央处理器(central processing unit，CPU)；存储器，包括随机存储器(random access memory，RAM) 和只读存储器(read only memory，ROM)；输入输出(I/O)接口。单片机指的是把这3部分做在一片集成电路芯片上。单片机一词最早出现在20世纪70年代后期，后来在国际上逐渐被微控制器(microcontroller 或 MCU)一词所替代。微控制器一词比单片机更简捷、更确切地描述了事物的本质，所以单片机一词在国际上已用得很少。而汉语中，单片机的叫法更通俗，以至一些学术机构，例如单片机学会采用了单片机的叫法，一时不宜更改。为同国际上的叫法一致，本书还是称之为微控制器。

现代化离不开微控制器，人民生活也离不开微控制器。在日常生活方面，洗衣机、电冰箱、空调器、电饭煲、电视机、音响、影碟机、照相机、游戏机、电话机、BP机内部都至少有一片微控制器。就个人计算机而言，它的主 CPU 是486、586…而它的键盘、鼠标器等里面都有一片微控制器。便携机的液晶屏、普通 PC 机的彩色监视器里面也都有一片微控制器。

在汽车上，除发动机的控制要用微控制器外，电动门窗、可升降座椅、安全空气囊、防抱死系统(ABS)、防盗报警器、可移动通信设备等都要用到微控制器。

在办公自动化方面，复印机、打印机、绘图仪、传真机等都是微控制器应用的例子。

在工业界，智能化仪表，机器人，生产过程的自动控制，任何温度、压力、流量的测量与控制，都离不开微控制器。

医疗仪器与机械、农业、化工、军事、航天等各个领域，也都离不开微控制器。

Motorola 公司是世界上最大的微控制器供应商。表1-1列出了1999年世界排名前10位的微控制器生产商的市场份额。1999年微控制器芯片的产值高达118亿美元。

表1-1　1999年世界上排名前10名的微控制器供应商市场份额

生产厂商	微控制器产值(百万美元)	占世界市场份额
Motorola	2006	17%
Hitachi	1419	12%
NEC	1341	11.4%
Mitsubishi	1005	8.5%

续表

生产厂商	微控制器产值(百万美元)	占世界市场份额
Fujitsu	660	5.6%
Toshiba	611	5.2%
Infineon Technologies	562	4.8%
Mitsubishi	545	4.6%
STMicroelectronics	545	4.6%
Texas Instruments	470	4%
其他厂商	2642	22.3%

表 1-2 给出了 1999 年世界排名前 10 名的 8 位微控制器供应商的微控制器销量。世界总销量约 34 亿片。

目前亚太地区微控制器用量与美洲市场、欧洲市场相比，相对份额较小，由于中国以及亚太地区其他发展中国家和地区经济的长期稳定高速发展，亚太地区在世界上所占有的地位将越来越重要。图 1-1 是 Dataquest 公司对亚太地区未来 5 年微控制器市场的预测。5 年以后，亚太地区对微控制器的需求将翻一番。

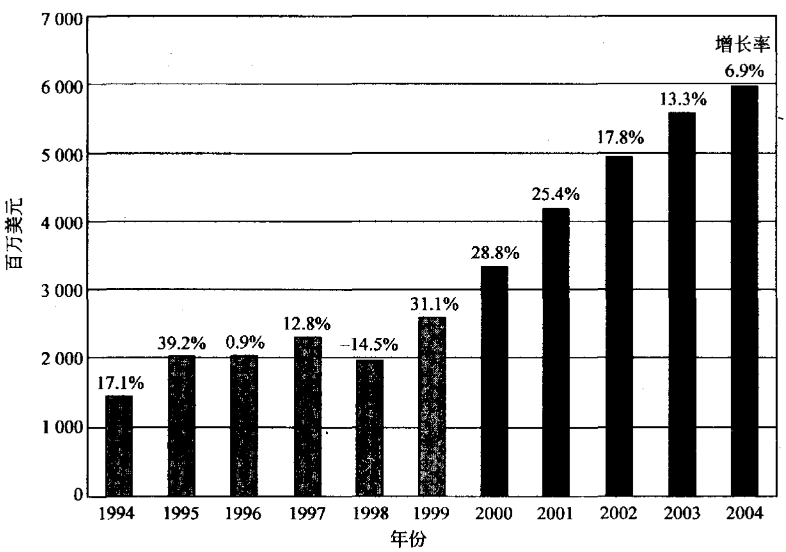


图 1-1　Dataquest 公司 2000 年 7 月对亚太地区未来 5 年微控制器市场的预测

表 1-2　1999 年全世界主要厂商 8 位微控制器的销量

生产厂商	8 位微控制器销量(片)	市场份额
Motorola	6.56 亿	19.3%
Microchip	3.065 亿	9%
Hitachi	3 亿	8.8%

续表

生产厂商	8位微控制器销量(片)	市场份额
NEC	2.4亿	7.1%
Infineon Technologies	2.17亿	6.4%
STMicroelectronics	1.91584亿	5.6%
Mitsubishi	1.9亿	5.6%
Philips	1.89474亿	5.6%
Toshiba	1.3亿	3.8%
Zilog	1.20187亿	3.5%
其他厂商	8.51290亿	25.3%

1.2 微控制器与微处理器——微计算机技术的两大分支

微处理器(microprocessing unit,MPU)是计算机的核心部件,用来制造计算机。计算机系统的主要用途是科学计算、数据处理、图象分析、数据库、人工智能、数字模拟与仿真等。

微控制器则主要用于控制目的。对以微控制器构成的系统的要求是有实时、快速的外部响应,能迅速采集到大量数据,作出逻辑判断与推理后实现对被控制对象的参数调整与控制。

显然,这两方面的要求是很不一样的。如果要求每一种微计算机都兼顾这两方面的要求,则会限制整个学科的发展。事实上,微计算机技术形成两大分支后在各自独立发展的道路上都得到了突飞猛进的发展。以 IBM-PC 为例,十余年来,微处理器从 8088、286、386、486 发展到 586、奔腾等。8 位机、16 位机都已经淘汰。32 位机也以每两、三年换代一次的速度向 64 位机发展。微处理器的发展动力在于人类对无止境的海量数值运算的需求。它是计算机科学与微电子技术学科高速发展的结果。计算机系统速度越来越快,功能越来越强,而对用户的要求则越来越趋向于“傻瓜”化。

随着微处理器的发展,微控制器虽然也发展出 16 位、32 位,但目前乃至今后相当长的时期内仍将以 8 位机为主。从年产量看,8 位微控制器的世界年产量已经超过了 10 亿片。以 Motorola 683xx 系列及 Intel 386 EX 为代表的 32 位微控制器,虽然有广阔的应用前景,但在数量上不可能达到年产量 10 亿片的水平。增强型 8 位微控制器以 MC68HC11 为代表,性能上已能与 16 位机媲美,而 32 位微控制器以其功能超群、开发容易、价格并不太高为特点,使 16 位微控制器的生存空间有限。未来的微控制器仍将以 8 位机为主、以 32 位为辅进行发展。微机技术在家用电器、智能化仪表、自动控制等方面的应用驱动微控制器的发展,几乎没有哪个行业用不着微控制器。微控制器的发展离不开各行各业的技术人员。微控制器直接使用了微处理器的成果,提高了构成系统的可靠性和抗干扰能力,降低了功耗与造价。

微控制器的发展是为了满足被控制对象的要求，构成各种专用控制器与多机控制系统。此外，它的发展还体现在：传感器接口，各种工业对象的电气接口，伺服驱动的功率接口，人机对话接口，通信网络接口等方面。这些接口性能的发展体现在：高速 I/O 能力，中断处理能力，A/D、D/A 的速度和精度，位操作能力，功率驱动能力，程序运行监控能力，信号实时处理能力等。

随着微处理器的发展，不久的将来也会出现 64 位微控制器，但微控制器总的发展方向仍将以 8 位为主，向高性能、高可靠性、低电压、低功耗、低噪声、低成本的方向发展。

1.3 微控制器与专用集成电路

ASIC(application specified integrated circuit)在我国被称作专用集成电路，指为某种用途专门设计制造的集成电路。其中所谓半定制 ASIC 设计指用户根据半导体集成电路制造商提供的单元电路库来设计自己的 ASIC。把微处理器看成电路库元件中的一个标准单元，微控制器就成了 ASIC。

Motorola 公司把 8 位微控制器，如 MC68HC08 归类于 CSIC (customer specified integrated circuit)，意为用户定义的集成电路；把 MC68HC08 微处理器看作库元件中的一个。库元件还有各类存储器，如 RAM、ROM、EPROM、EEPROM、Flash(闪速存储器)等不同容量的模块，以及各类 I/O 模块，如并行口、串行口、定时器、A/D 变换器、脉宽调制器、DMA(存储器直接存取)控制器、网络通信控制器等。用户可根据自己应用上的需要，任选若干模块构成自己的专用集成电路。即使选用手册上的通用微控制器，当厂家把用户开发好的程序固化到微控制器中的掩膜 ROM 中去以后，也成了该用户的专用电路。厂家可以按照用户的要求在这种集成电路上印上用户命名的型号。典型微控制器的价格与数量的关系如图 1-2 所示。

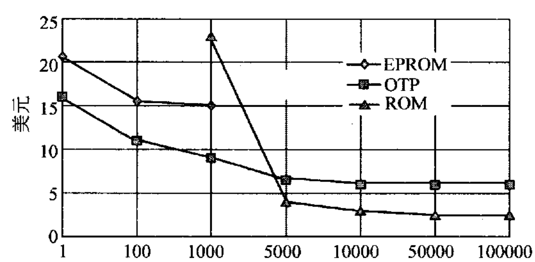


图 1-2 典型 MCU 的价格与数量的关系

同设计 ASIC 一样，这种做法的前提是要有一定的批量才可接受。目前，这个批量概念的数量在 30000 以上。换句话说，一个产品有足够大的批量，一种新型的微控制器就出现了。用这种方法设计 ASIC，即直接利用现成的库元件，比从门电路设计起的方法成本低，风险小。

我国是一个发展中国家，在 ASIC 设计方面比较落后，新产品开发中可注入的资金有限。用现成的微控制器构成自己专用的集成电路就显得特别有意义。只要产品有 3000

以上的批量，就可以请厂家把用户开发的应用程序固化到微控制器中去，从而形成自己的专用集成电路。这种批量生产的固化程序的工艺俗称掩膜。6805 类一般性微控制器典型价格在 2～3 美元之间，简单的、封装引脚少、批量大的微控制器单价已降低至 1 美元。

当产品批量小时，可使用 OTP(one time programable，意为一次性可编程的)型微控制器程序由用户自行写入，目前 OTP 型微控制器平均价格为掩膜型的 2.5～3 倍。

在产品开发初期，也可以选用 EPROM 型微控制器。程序可以写入，也可以用紫外灯照射擦除，反复使用。EPROM 型微控制器从外观上看有一个透紫外光的玻璃窗或塑料窗。

近年来，闪速存储器 Flash 在微控制器片内的应用走向成熟。Flash 的片内制造工艺、Flash 的保护技术有突破性进展；Flash 在遇到强干扰引起程序丢失的问题得到了妥善的解决，从而引发微控制器开发、应用技术的又一次飞跃。Flash 是一种非易失性储存介质，读取 Flash 中的内容同 RAM 读取没有什么不同。Flash 的写操作也比 EPROM 要快。而系统掉电以后，Flash 中的内容仍可以可靠地保持不变。Flash 的另一大优点是结构简单、集成密度大、成本低。形象地理解 Flash 的结构，存储器的一位使用一只晶体管就够了，而 RAM 中的一位至少要 6 只晶体管。由于 Flash 比 EPROM 便宜，不需要用紫外灯照射法擦除，CPU 可以依靠程序擦除 Flash 的某一部分或全部，反复使用次数可达数万至数十万次，有替代 EPROM 的趋势。比起掩膜 ROM 来，Flash 也有相当大的优势。首先是 8 到 16 周的掩膜芯片生产周期省去了，掩膜费用节省了。其次是掩膜后程序不可再修改，在采用 Flash 后掩膜法的这种风险没有了。另外，程序可以在产品出厂前灌入 Flash，也可以在产品出厂后，将应用程序灌入或升级。

Flash 的使用还带来了微控制器开发手段的革命，由于 Flash 可以局部擦除，且写入、擦除次数可达数万次以上，开发微控制器不再需要昂贵的仿真器。可以在微控制器的 Flash 中先灌入一个用于应用程序调试的监控程序，依靠这个监控程序的帮助，使用一次又一次地灌入、擦除被调试程序的方法调试应用程序。用监控程序法调试应用程序将在本书的后几章中详细介绍。

1.4 微控制器的历史与发展

20 世纪 70 年代后期，4 位逻辑控制器件发展到 8 位。当时可称得上是 8 位微控制器的应以 MC6801 为代表，它以当时流行的微处理器 MC6800 为 CPU，使用 NMOS 工艺将并行口、定时器和 128 字节 RAM 集成在一个芯片内。Intel 同期的代表产品是 8048，可称之为第一代微控制器。

最早采用 CMOS 工艺的微控制器是 Motorola 的 MC146805，出现在 20 世纪 80 年代。1982 年以后，NMOS 工艺逐渐被高速低功耗的 HCMOS 工艺代替。HCMOS 是高速互补型金属氧化膜半导体的英文字头。MC68HC05 最早出现于 1982 年，十多年来，已由当时的一种发展到如今的数百种。Intel 8051 系列微控制器进入中国市场最早，过渡到 CMOS 工艺的 80C51 相对晚些。近年来 Philips 等公司开始生产与 8051 兼容的微控制器，扩展了其功能。以早期的 MC146805 与 Intel 8051 为代表的微控制器可称为第二

代微控制器。

近十年来，微控制器的发展出现了许多新特点，所以可称之为第三代微控制器。

在技术上，由可扩展总线型向纯单片型发展，即只能工作在单片方式。内存容量已做得相当大，I/O 功能已足够丰富，不需要外加扩展芯片。这样既降低了整机成本，又可以防止伪造。掩膜产品价格可接受的起点降至 2000 片到 3000 片。

微控制器的扩展方式从并行总线型发展出各种串行总线，并被工业界接受，形成一些工业标准，如 I^2C 总线、SPI 总线。甚至将网络协议的低两层或低三层协议都集成进去，如 CAN 总线。

将多个 CPU 集成到 1 个微控制器中去已不是什么新鲜事，例如 MC143120/MC143150 中集成有 3 个 CPU，一个用于 I/O 控制；一个用于类似以太网协议的通信；一个用于总体协调。随着超大规模集成电路技术的发展，将数字信号处理器、精简指令流型微处理器等集成到微控制器中去的产品已经出现。

Flash 的使用使微控制器技术进入了第四代。

在降低功耗、提高可靠性方面，微控制器工作电压已降至 3.3V。工作电压为 2.7V、1.8V 的微控制器也已经出现。有的微控制器的工作电压将降至 0.9V。抗噪声、抗干扰方面也不断出现一些新技术。

一些小公司也不断推出它们的微控制器产品。这些公司的微控制器许多是为它们本公司的电气产品设计的。它们原来只不过是微控制器的用户。由于用户的介入，微控制器的控制功能得到了飞速的发展。这些功能的发展，远远超出了计算机科学的领域，应用领域之广几乎是无限的。小到信用卡、玩具，大到航天器、机器人。实现数据采集、过程控制、模糊控制等智能系统，都离不开微控制器。

大的电气厂商也结合自己的产品形成自己的微控制器体系。各类微控制器优势互补自寻最佳应用领域，必将形成百花齐放、竞争激烈的局面。

1.5 微控制器的开发手段

把程序固化到微控制器内部有很多好处。既提高了系统的抗干扰能力，提高了产品的可靠性，又降低了成本，降低了功耗，使自己的产品不易被仿造。然而同时也给产品的开发工作带来了一定的难度。

简单的微控制器系统，可以采用 EPROM 型微控制器，用反复写 EPROM 的方法来开发。这要求开发者对该类微控制器特别熟悉，且系统比较简单。

使用仿真器是开发微控制器的一种方法，特别是对只有单片工作方式的微控制器(即总线不可扩展型微控制器)，仿真器价格在 350 到 700 美元之间。先将 PC 机串行口与仿真器串行口相连，接着将仿真器伸出一只与微控制器引脚数目相同的电缆与插头，即仿真头，插到被开发的系统微控制器所在位置上。在 PC 机上编程，然后将程序下载到仿真器上。通过仿真头调试目标板，调试完成后，将程序写到一片微控制器中去，把仿真头换下来。一套仿真器，一般可仿真一到两个系列的微控制器。MC68HC05 有几十个系列，所以为每一系列微控制器都购置一套仿真器显然是不能承受的。

总线不可扩展型微控制器的另一种开发方法是：先使用总线可扩展的微控制器，将EPROM扩展到微控制器外面来。微控制器内部没有的I/O功能模块也用扩展方式加上去，在样机研制完成以后再选用不可扩展型的微控制器做出产品。把EPROM扩展到外面来以后开发工作就容易做了。一个办法是先写一个监控程序，建立一个调试环境，直接在目标板上调程序。另一个办法是使用EPROM仿真器，使PC机上写好的程序可以一次次装到“EPROM”中进行调试。

在Flash技术用于微控制器片内以后，监控程序法已不需要在总线外面扩以EPROM。而Flash法和上述监控程序法在原理上没有什么不同，只是更方便、更快捷、更便宜。本书将重点介绍用监控程序法调试Flash中的应用程序。

还有一种开发微控制器的办法是在线非实时仿真法。用比被开发微控制器多一个口（串行口或并行口的几位）的微控制器，做成一种廉价的开发工具。用这个多出来的口与PC机的串口或并口相连，实现微控制器与PC机的通信，即实现人与微控制器的对话。被调试程序与接口有关的部分即I/O部分用微控制器实现。其他软件部分在PC机上用PC机的CPU以在线非实时仿真完成。由于PC机的CPU与8位微控制器的CPU相比要强有力得多，所以用PC机对微控制器的程序作出解释后在PC机上模拟运行，只在微控制器上做与I/O相关的操作。这种开发办法适用于较简单的微控制器。例如，用MC68HC05J2开发MC68HC05K1，用MC68HC05C8开发MC68HC05J2 。这种开发工具价格很便宜。

1.6 Motorola 微控制器命名法

下面以MC68HC708XL36为例说明Motorola微控制器命名法，如图1-3所示。

MC	68	HC	7	08	XL	36	XX	B/FU
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)

图1-3　Motorola MCU 命名法

图1-3中各符号和数字的含义为

(1) PC—质量不保证，用于测试的器件；MC—质量保证的正规产品；XC—有限质量保证，用于性能评估的器件。

(2) 由于历史原因，Motorola的微控制器类产品都冠以68。

(3) 速度/供电电压：H—高速；C—CMOS；L—低电压。

(4) 片内存储器类型：没有—掩膜型；7—EPROM/OTP；8—EEPROM；9—Flash。

(5) CPU类型：08。

(6) 系列及系列号：XL—系列号。含义见表1-3。

(7) ROM容量：36～36kB ROM。

(8) 适用温度范围：没有—0～70℃；C—0～85℃；M—－40～125℃。

(9) 封装形式。

表 1-3　系列号中相关字符的含义列表

相关字	含　　义
A	汽车类
B	通用型 16/20 引脚
C	消费类产品
D	DSP
E	通用型
F	电话类
G	通用类
H	微机辅助芯片
I	不用
J	通用型微机外设
K	模糊控制类
L	液晶
M	马达控制
N	通用型 52/56 引脚
O	不用
P	PC 类
Q	保留给将来使用
R	保留给将来使用
S	智能卡类
T	电视类
U	不用
V	真空荧光显示器控制类
W	保留给将来使用
X	通用型,带 CAN 总线
Y	通用型 80 引脚
Z	通用型 100 至 132 引脚

MC68HC08 的前身是 MC68HC05,它出现于 20 世纪 80 年代中期。MC68HC05 家族目前仍是世界上品种最多、产量最高的微控制器。以 MC68HC05 为 CPU 的微控制器有近 50 个家族,300 多个品种。MC68HC08 是 MC68HC05 的升级换代产品,目前的品种还不算多。表 1-4 中列出了目前已上市的 15 种微控制器。不久的将来,会有更多的 MC68HC08 家族产品推出。2001 年会推出 MQ 系列、GR 系列、RF 系列、RK 系列,以后如驱动液晶的 L 系列、用于电视机的 T 系列等也将问世。

表 1-4　MC68HC08 家族

产品	ROM	RAM	Flash	EPROM /OPT	EEPROM	定时器	I/O	串口	A/D	PWM	工作电压 /V	总线频率 /MHz
MC68HC08AZ0		1kB			512B	定时器 1:4 通道 定时器 2:2 通道 16 位的 IC,OC 或 PWM	48	SCI SPI	8 通道、8 位	定时器 1:4 通道 定时器 2:2 通道 16 位 PWM	5.0	8.0
MC68HC08AZ32	32kB	1kB			512B	定时器 1:4 通道 定时器 2:2 通道 16 位的 IC,OC 或 PWM	48	SCI SPI	8 通道、8 位	定时器 1:4 通道 定时器 2:2 通道 16 位 PWM	5.0	8.0
MC68HC08LN56	56kB	1280B 及 LCD 的 16B				定时器 1:4 通道 16 位的 IC,OC 或 PWM	42	SCI SPI	4 通道、8 位	定时器 1:4 通道 16 位 PWM	3.3、5.0	8.0
MC68HC908AZ60		2kB	60kB		1B	定时器 1:6 通道 定时器 2:2 通道 16 位的 IC,OC 或 PWM	48	SCI SPI	15 通道、8 位	定时器 1:6 通道 定时器 2:2 通道 16 位 PWM	5.0	8.0
MC68HC908GP20		512B	20kB			定时器 1:2 通道 定时器 2:2 通道 16 位的 IC,OC 或 PWM	33	SCI SPI	8 通道、8 位	定时器 1:2 通道 定时器 2:2 通道 16 位 PWM	3.0、5.0	8.0
MC68HC908GP32		512B	32kB			定时器 1:2 通道 定时器 2:2 通道 16 位的 IC,OC 或 PWM	33	SCI SPI	8 通道、8 位	定时器 1:2 通道 定时器 2:2 通道 16 位 PWM	3.0、5.0	8.0
MC68HC908JK1		128B	15kB			定时器 1:2 通道 16 位的 IC,OC 或 PWM	15		10 通道、8 位	定时器 1:2 通道 16 位 PWM	3.0、5.0	8.0
MC68HC908JK3		128B	4kB			定时器 1:2 通道 16 位的 IC,OC 或 PWM	15		10 通道、8 位	定时器 1:2 通道 16 位 PWM	3.0、5.0	8.0
MC68HC908JL3		128B	4kB			定时器 1:2 通道 16 位的 IC,OC 或 PWM	23		12 通道、8 位	定时器 1:2 通道 16 位 PWM	3.0、5.0	8.0

续表

产品	ROM	RAM	Flash	EPROM /OPT	EEPROM	定时器	I/O	串口	A/D	PWM	工作电压 /V	总线频率 /MHz
MC68HC708MP16		512B		16kB		定时器 1:4 通道 定时器 2:2 通道 16 位的 IC,OC 或 PWM	44	SCI SPI	10 通道、8 位	定时器 1:4 通道、16 位 PWM 定时器 2:2 通道、16 位 PWM 6 通道、12 位 PWM	5.0	8.0
MC68HC08MR4		192B				定时器 1:2 通道 定时器 2:2 通道 16 位	22	SCI	4 通道或 7 通道、8 位	6 通道、12 位 PWM	5.0	8.0
MC68HC908MR8		256B	8kB			定时器 1:2 通道 定时器 2:2 通道 16 位	22	SCI	4 通道或 7 通道、10 位	6 通道、12 位 PWM	5.0	8.0
MC68HC908MR16		768B	16kB			定时器 1:4 通道 定时器 2:2 通道 16 位的 IC,OC 或 PWM	44	SCI SPI	10 通道、10 位	6 通道、12 位 PWM	5.0	8.0
MC68HC908MR24		512B	24kB			定时器 1:4 通道 定时器 2:2 通道 16 位的 IC,OC 或 PWM	44	SCI SPI	10 通道、10 位	定时器 1:4 通道、16 位 PWM 定时器 2:2 通道、16 位 PWM 6 通道、12 位 PWM	5.0	8.0

注：OTP(one time programable)——次性可编程；

IC(input capture)—输入捕捉；

OC(output compare)—输出比较；

PWM(pulse width modulation)—脉宽调制；

SCI(serial communications interface)—串行通信接口；

SPI(serial peripheral interface)—串行外设接口。

第 2 章　新一代 8 位微控制器 MC68HC08

正如第 1 章中所介绍的，Motorola 8 位微控制器 MC68HC05 从出现至今已有近 20 年的历史，具有几十个系列、几百个品种，应用范围极其广泛。

随着微电子技术的飞速发展，Motorola 将 Flash 及锁相环等技术引入微控制器，于 1999 年推出新一代高档 8 位微控制器 MC68HC08 系列产品。MC68HC08 以其功能强、成本低、功耗低、开发容易等优点将逐步取代 MC68HC05，成为今后 8 位微控制器应用领域的主流机型。本章以 MC68HC908GP32 为例简单介绍 MC68HC08 系列微控制器的结构、特性、功能和管脚，详细介绍了它的存储器组织和 CPU 结构。

2.1　MC68HC08 系列微控制器概述

2.1.1　MC68HC08 各系列简介

Motorola 公司的 8 位微控制器的产量一直居世界第一位，约占整个国际市场的 1/3 左右。1999 年推出的新一代 8 位微控制器——MC68HC08 采用了 0.35μm 工艺，具有速度快（8MHz 总线速度）、功能强、功耗小及价格低等优点，特别是带有闪速存储器 Flash 的 MC68HC908 具有更高的性能价格比。图 2-1 是 Motorola 公司 MC68HC08 系列微控制器产品发展图。

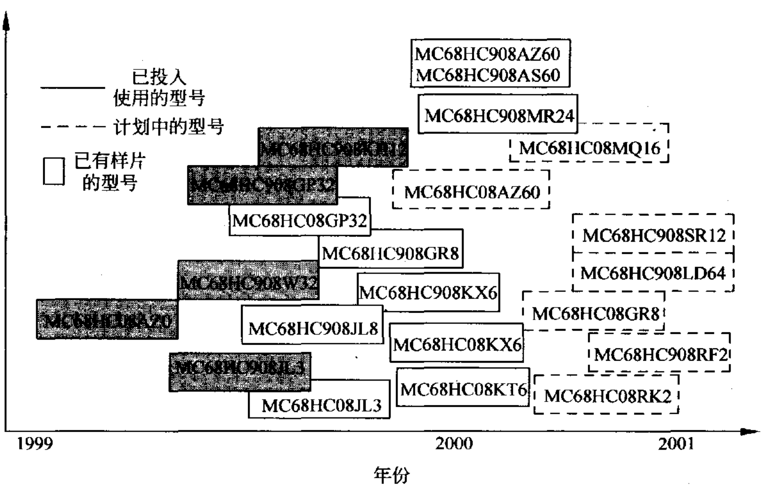


图 2-1　MC86HC08 系列微控制器产品发展图

从图 2-1 可以看出 MC68HC08 正在不断增加新产品，有通用的 GP、JL 和 XL 型，汽

车控制的 AZ 型，模糊控制的 KX、KJ 型，马达控制的 MR 型，电话用的 W 型等各种微控制器，以后还将推出电话控制型、家用消费型、智能 IC 卡型、LCD 驱动及 VFD 驱动型微控制器，MC68HC08 将会逐步覆盖 MC68HC05 的应用范畴。此外，还将推出内部带 MC68HC08 和数字信号处理器的多 CPU 系列产品。MC68HC08 虽然性能比 MC68HC05 性能高得多，而价格却不会高于原有的 MC68HC05。MC68HC08 系列微控制器是 Motorola 公司今后主要发展的 8 位微控制器。

2.1.2 MC68HC08 系列微控制器的主要特点

MC68HC08 系列微控制器的主要特点为

(1) 采用模块化设计，各种不同型号微控制器由不同模块组成，7 天就可以设计出用户所需的微控制器。

(2) 含片内监控 ROM，为用户提供了在线编程及在线调试等功能。

(3) 具有特色的 Flash 取代片内 EPROM 和 ROM，其价格低于相同容量的 OTP 型微控制器。

(4) 具有锁相环电路，可以使用 32kHz 的晶振产生 8MHz 的总线速度，大大降低了干扰。

(5) 与 MC68HC05 向上兼容，不同之处主要是：

——变址寄存器由 8 位变为 16 位；

——堆栈指针 SP 由 6 位变为 16 位；

——程序计数器 PC 也为 16 位；

——增加 8 种寻址方式和 78 条指令。

2.1.3 MC68HC08 中央处理器 CPU08

1. CPU08 的结构

MC68HC08 微控制器皆以 8 位的 CPU08 为中央处理器。CPU08 是与 CPU05 指令代码向上兼容的 CPU，但性能更好、速度更快。

CPU08 由 3 个部分组成，如图 2-2 所示。

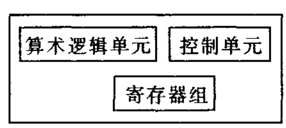


图 2-2　CPU08 的结构

2. CPU08 的特性

CPU08 的主要特性为

(1) 与 CPU05 指令代码完全向上兼容；

(2) 4kB 程序/数据存储器空间；

(3) 8MHz CPU 内部总线频率；

(4) 16 种寻址方式，相对于 CPU05 增加了 8 种；

(5) 可扩展的内部总线定义，用于寻址超过 64kB 的地址空间；

(6) 用于指令操作的 16 位变址寄存器；

(7) 16 位堆栈指针和相应栈操作指令；

(8) 不使用累加器的存储器之间的数据移动；

(9) 快速 8 位乘法和 16 位除法指令；

(10) BCD 码指令进一步增强；

(11) 增强型外设，如 DMA 控制器；

(12) 完全的静态低电压/低功耗设计。

3. CPU08 的寄存器

CPU08 内部寄存器如图 2-3 所示，包括如下 5 个部分。

(1) 累加器

累加器 A 是通用 8 位寄存器。CPU 用 A 保存操作数及运行结果。

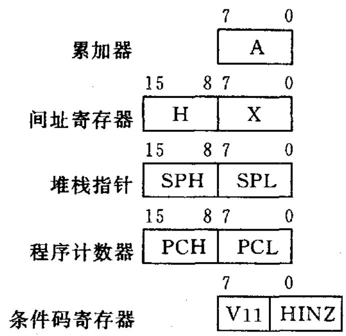


图 2-3 CPU08 内部寄存器

(2) 变址寄存器

变址寄存器 H∶X 是个 16 位寄存器，高 8 位用 H 表示，低 8 位用 X 表示。复位清零 H。当 H=0 并且没有使用影响 H 的指令时，H∶X 的功能与 CPU05 的 X 寄存器是相同的。CPU08 一般用 H∶X 的内容表示操作数的地址，但 H∶X 也可以暂时用于储存数据。H 或 X 可以暂存 8 位数据，而 H∶X 可以暂存 16 位数据。

(3) 堆栈指针

堆栈指针(stack point，SP)是个 16 位寄存器，复位时被置为 $00FF，与 CPU05 相同。RSP 指令使 SP 的低 8 位为 $FF，而高 8 位不受影响。数据入栈时，SP 减小；数据出栈时，SP 增加，SP 永远指向下一个可用的(空的)单元。尽管 SP 被复位为 $00FF，但实际上堆栈的位置是任意的，并可以由用户将之定义在 RAM 中的任意位置上，若将 SP 移出第 0 页($0000～$00FF)，便可得到更多的可以使用直接寻址方式的空间。

(4) 程序计数器

程序计数器(program counter，PC)是个 16 位寄存器，它的内容表示下一条指令或下一个操作数的地址。复位时，PC 被置为复位向量地址 $FFFE。$FFFE 和 $FFFF 单元的内容，即为复位后要执行的第一条指令的地址。

(5) 条件码寄存器

条件码寄存器(condition code register，CCR)如图 2-4 所示。

位	7	6	5	4	3	2	1	0	
	V	1	1	H	I	N	Z	C	CCR
复位	×	1	1	×	1	×	×	×	

图 2-4 MC68HC08 的条件码寄存器

CCR 包含一个控制位(中断屏蔽位 I)和 5 个记录指令执行结果特征的标志位,第 5、6 位永远为 1。

① V——溢出标志

有符号跳转指令 BGT、BGE、BLE 和 BLT 使用该标志。

1=二进制补码有溢出。

0=二进制补码无溢出。

② H——半进位标志

BCD 码运算(DAA 指令)需要使用 H(和 C)标志。

1=执行 ADD 和 ADC 指令时,累加器第 3 位向第 4 位有进位。

0=执行 ADD 和 ADC 指令时,累加器第 3 位向第 4 位无进位。

③ I——中断屏蔽标志

中断屏蔽标志 I 是个控制位,使用指令 SEI 及 CLI 可以使之置 1 或置 0。

当该位置 1 时,所有可屏蔽中断都被禁止。复位时,该位置 1。当用 CLI 指令使该位置 0 时,CPU 中断得到允许。中断响应时,CPU 将除 H 以外的寄存器推入堆栈,以保护断点和现场,然后执行中断服务子程序;遇到 RTI 指令时,从栈中恢复包括 CCR 在内(当然也包括这一位的状态)的各寄存器,以恢复断点和恢复现场。注意:在中断服务子程序中如用到 H 寄存器的话,不要忘了使用 PUSH 指令保存 H 的内容和使用 PULH 指令恢复 H 的内容。

1=中断禁止。

0=中断允许。

④ N——负标志

1=运算结果为负(最高位为 1)。

0=运算结果为正(最高位为 0)。

⑤ Z——零标志

1=数据或运算结果为 0。

0=数据或运算结果非 0。

⑥ C——进位/借位标志

1=最高位上有进位或有借位。

0=最高位上无进位或无借位。

从图 2-5 可以清楚地看出 CPU05 寄存器与 CPU08 寄存器的区别。CPU08 增加了变址寄存器 H、溢出标志位 V;堆栈指针 SP 由 CPU05 的 6 位增至 CPU08 的 16 位,而且可以编程。CPU08 堆栈空间大,而且可以重新定位于 64kB 空间中的任意位置上,这就使 CPU08 具有更加灵活多变的寻址方式、更丰富的汇编指令系统,并且可以使用 C 语言编程。详细内容请查阅有关章节。

虽然 CPU08 内部寄存器较少,但由于片内存储器第 0 页含 64B I/O 寄存器和 192B RAM($40~$FF),都可以用直接寻址方式实现数据从存储器到存储器的直接传送,即不必经过累加器 A。也就是说第 0 页存储单元都可以当作寄存器用。而另外 320B RAM 也可以用间接寻址的方式实现存储器到存储器的数据传送。因此可以认为 CPU08 的寄

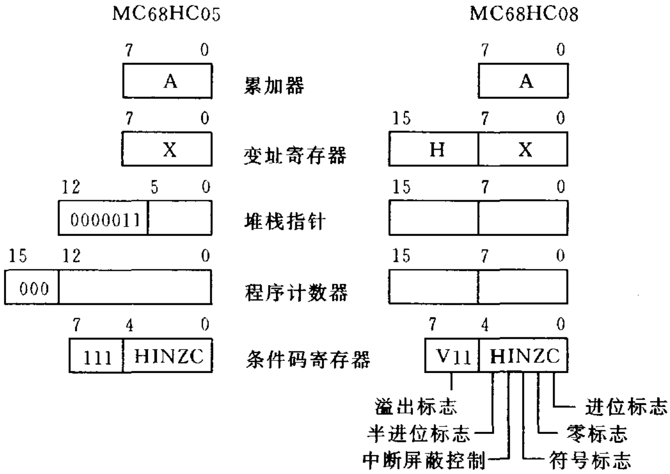


图 2-5　CPU08 与 CPU05 寄存器的比较

存器相当多。把内部存储器当作寄存器使用可大大提高代码效率，而内部寄存器少又使得中断响应速度提高。这种设计思想体现了微控制器与微处理器面向不同应用领域的思想。微控制器更强调面向控制。

4. CPU08 的总线时钟与时序

MC68HC08 通常工作在单片方式，不需要甚至不可能在 CPU 总线上扩展外围器件。CPU 的内部时序对开发人员来说并不重要。然而了解 CPU08 的总线时钟与时序，对于理解 CPU 的工作过程以及详细计算程序执行时间是非常有帮助的。

典型的 CPU08 总线时钟为 8MHz，可以理解为 CPU 作一次总线读操作的时间是 125ns(写操作也是一样)。内部地址总线每次切换到下一个地址的时间是 125ns，内部数据总线上数据的稳定时间也是 125ns，即 CPU 以 8MHz 的总线频率一拍一拍地运行着。仔细分析运行的过程可知，CPU 要从存储器中读取指令，对指令作出解释，然后才执行指令。而读取存储器中的指令则要先将有效地址(一般在 PC 寄存器中)送到地址总线上去，经过一段时间地址稳定了，存储器再将操作码或操作数送到数据总线上，又经过一段时间数据稳定了，CPU 才能将数据读进来。这一过程如图 2-6 所示。

原来，CPU08 的总线时钟周期由 $T1$、$T2$、$T3$、$T4$ 共 4 个状态组成，每个状态的相位相差 90°(为便于理解，假设每个状态只有上升沿起作用)。这就是为什么要想得到 8MHz 的总线频率，锁相环时钟产生器的频率要编程为 32MHz，即总线频率的 4 倍的原因。从图 2-6 中可以看出，数据总线上数据的更新是在 $T1$ 的上升沿开始的，而地址总线上下一个新地址则是在 $T3$ 的上升沿作用下发生的。$T2$ 和 $T4$ 这两个状态则用于指令的读入和执行。

地址总线、数据总线是 CPU 用来同存储器打交道的。CPU 内部可以分为两部分，它们是有限状态机部分和执行单元部分。有限状态机负责对当前状态，并根据从存储器读回来的指令，给出 CPU 的下一状态应该做什么，然后将信息传给执行单元去执行。执行

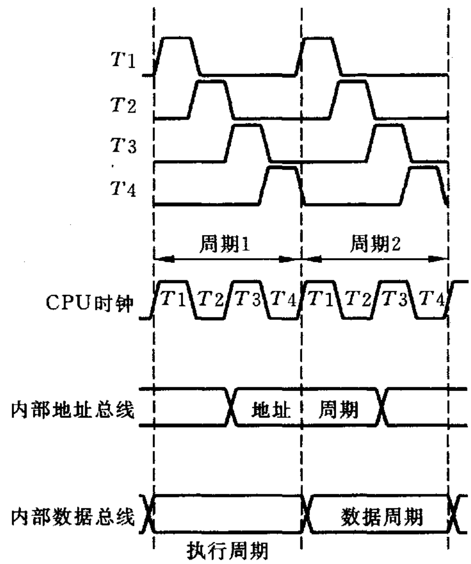


图 2-6　CPU08 内部时序详解

单元包括 CPU 中的内部寄存器,如 A、H:X、PC 等等。有限状态机的运行领先于执行单元约一个总线时钟周期,执行单元则在以后的时钟周期内完成地址总线、数据总线的操作。从图 2-7 CPU08 内部控制时序可以看出,一个读指周期或一个指令执行周期要延续 3 个总线时钟周期即 12 个 T 状态,也就是大约 375ns,并非 1 个总线周期 125ns。但由于有限状态机、执行单元、地址、数据总线的运行是并行的,执行单元在执行指令时,有限状态机已经在读下一条指令了,故看起来一个读指令的周期只有 125ns。

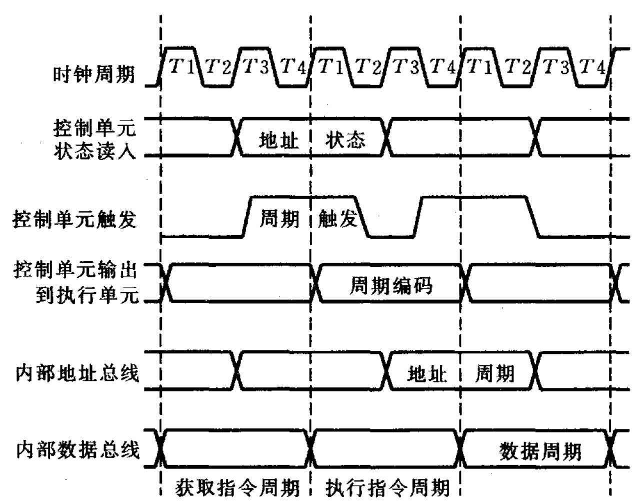


图 2-7　CPU08 内部控制时序

通过下述程序可进一步了解 CPU08 的内部时序和指令执行时间。

```
                ORG     $50
                FCB     $12,$34,$56
                ORG     $100
0100  A6    50  LDA     #$50        ;A=$50        PC=$0103
0102  97        TAX                 ;A→x          PC=$0104
0103  E6    02  LDA     2,X         ;[X+2]→A      PC=$0106
0105  5C        INCX                ;X=X+1        PC=$0107
0106  C7 80 00  STA     $8000       ;A→$8000      PC=$010A
```

图 2-8 是以上程序执行过程的时序图。图中可以看出，在 CPU08 内部执行 TAX，把 A 寄存器的值传给 X 的时候，下一条指令的有效地址 $0103 已经出现在总线上，而此时程序计数器 PC 中的值已经指向 2 字节指令的第二个字节 $0104 了。接着数据总线上出现存储单元 $0103 中的值 $E6，这是 CPU 读入 LDA 2,X 这个 2 字节指令的第一个字节。注意，此时 CPU 仍处在执行 TAX 这条指令的周期中，读入的 LDA 2,X 这条指令先

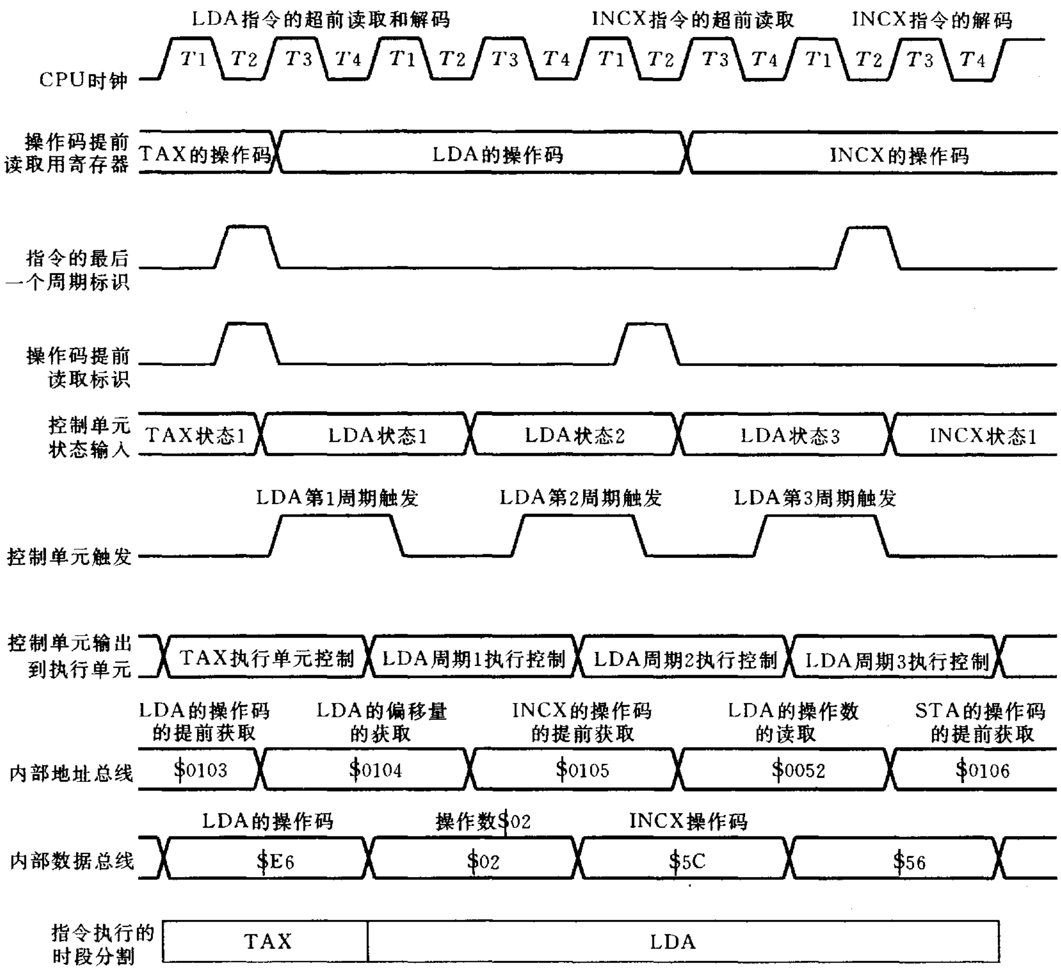


图 2-8 指令执行时序详解

进入超前指令寄存器，待 TAX 指令完成以后，通信进入 CPU 状态机，再进入执行机构。

同样如图 2-8 所示，LDA 2,X 是一条 2 字节、执行时间为 4 个周期的指令。前两个周期用于读入 \$0103 和 \$0104 这两个单元中的操作码和操作数 \$E6 和 \$02，第 3 个周期对这条指令作出解释，第 4 个周期完成这条指令。由图 2-8 可以看出，由于 X 的值是 \$50，X+2= \$52，而 \$52 单元中的内容是 \$56 ，LDA 2,X 把 \$56 单元的内容传给寄存器 A。于是在第 4 个周期中，地址总线上出现了地址 \$0052，数据总线上要传给 A 的内容是 \$56。注意，在第 3 个周期中，状态机对 LDA 2,X 这条指令进行解释的时候，下一条指令 INCX 的地址 \$0105 和操作码 \$5C 已经出现在地址总线和数据总线上了。即在 CPU 解释 LDA 2,X 这条指令的时候，下一条指令 INCX 已经进入了 CPU 的超前读入寄存器。

由指令表（见附录）中的指令执行时间我们会发现，TAX 这条指令在 CPU05 中是 2 个周期，而在 MC68HC08 中是 1 个周期；LDA 2,X 这条指令在 CPU05 中是 4 个周期，而在 CPU08 中是 3 个周期；INCX 指令在 CPU08 中是 1 个周期，也比 CPU05 少用 1 个周期。这是因为指令的读入周期和上一条指令的执行周期是同时发生的。或者，如 LDA 2,X 的执行那样，实际是 4 个周期，而读下一条指令的操作在第 3 个周期时就发生了，第 4 个周期执行 LDA 2,X 可以看成是下一条指令的执行周期，看起来像是 3 个周期。

通过以上分析可以看出，CPU08 采用了流水线作业、并行处理等设计思想，使指令执行时间较 CPU05 要快，所用的周期数少。由于典型的 CPU08 总线时钟是 8MHz，而 CPU05 的典型总线时钟为 2MHz，已经快了 4 倍，加上上述分析中使用的指令超前读入技术等，可以认为，CPU08 比 CPU05 至少快 5 倍。

2.2 MC68HC908GP32 概述

MC68HC908GP32 是 MC68HC08 微控制器的第一批产品，是一种通用型微控制器，本节对它作一简单介绍。

2.2.1 MC68HC908GP32 的特性

MC68HC908GP32 以它的高性能、低功耗、低价位在推出后的短短一年时间内就获得了广泛的应用。它的主要特性为

(1) 32kB 片内 Flash 存储器，具有在线编程能力和保密功能。

(2) 512B 片内 RAM。

(3) 8MHz 内部总线频率。

(4) 增强型串行通信口和串行外围接口。

(5) 两个 16 位双通道定时器接口模块，每个通道可选择为输入捕获、输出捕获和脉宽调制输出，其时钟可分别选为内部总线的 1、2、4、8、16、32 和 64 的分频值。

(6) 8 路 8 位 A/D 转换器。

(7) 系统保护特性，包括计算机工作正常(COP)复位；低电压检测复位，可选为 3V 或

5V 操作；非法指令码检测复位；非法地址检测复位。

(8) 时钟发生器模块，用 32kHz 晶振的锁相环电路，可产生各种工作频率。

(9) 33 根通用 I/O 脚，包括 26 根多功能 I/O 脚和 5 或 7 根专用 I/O 脚。

(10) PA、PC 和 PD 的输入口有可选择的上拉电阻。

(11) 所有口有 10mA 吸流和放流能力，PTC0～PTC4 有 15mA 吸流和放流能力(总体驱动电流应小于 150mA)。

(12) 带时钟预分频的定时基模块有 8 种周期性实时中断(1Hz、4Hz、16Hz、256Hz、512Hz、1024Hz、2048Hz 和 4096Hz)，可在 STOP 方式时使用外部 32kHz 晶振周期性唤醒 CPU。

(13) 8 位键盘唤醒口。

(14) 所有口有最高 5mA 输入电流保护功能。

(15) 具有 PDIP40、SDIP42 和 QFP44 封装形式。

(16) 支持 C 语言。

(17) 完全向上兼容 MC68HC05。

(18) WAIT、STOP 低功耗模式。

(19) 上电复位。

2.2.2 MC68HC908GP32 的功能结构

MC68HC08 系列微控制器从 1999 年推出后已研制出多个系列、品种，由模块的不同组合而成，本节只简单介绍 MC68HC908GP32 的各个模块，并将在后面的各个章节中分别给予详细介绍。MC68HC908GP32 功能结构框图如图 2-9 所示。

1. CPU 08

CPU08 是 MC68HC908GP32 的中央处理器。CPU08 是 CPU05 的增强型，但仍是 8 位 CPU。

2. 存储器

MC68HC908GP32 可寻址 64kB 地址空间。它主要包括

(1) 32kB 的闪速存储器 Flash；

(2) 32256B 的用户空间；

(3) 512B 的随机存储器 RAM；

(4) 36B 用户定义的矢量区(Flash 存储器)；

(5) 307B 的监控 ROM。

Flash 存储器是一种快速、非易失、在高压下进行擦写的存储器。因为 MC68HC908GP32 有在片的电荷泵可以产生 Flash 擦写所需要的高压，所以其芯片只需要单一的外部电源就可实现 Flash 的读、写、擦除的全部操作。MC68HC908GP32 内部有 32kB Flash 存储器，其写入与擦除主要由 FLCR 寄存器($FE08)控制。

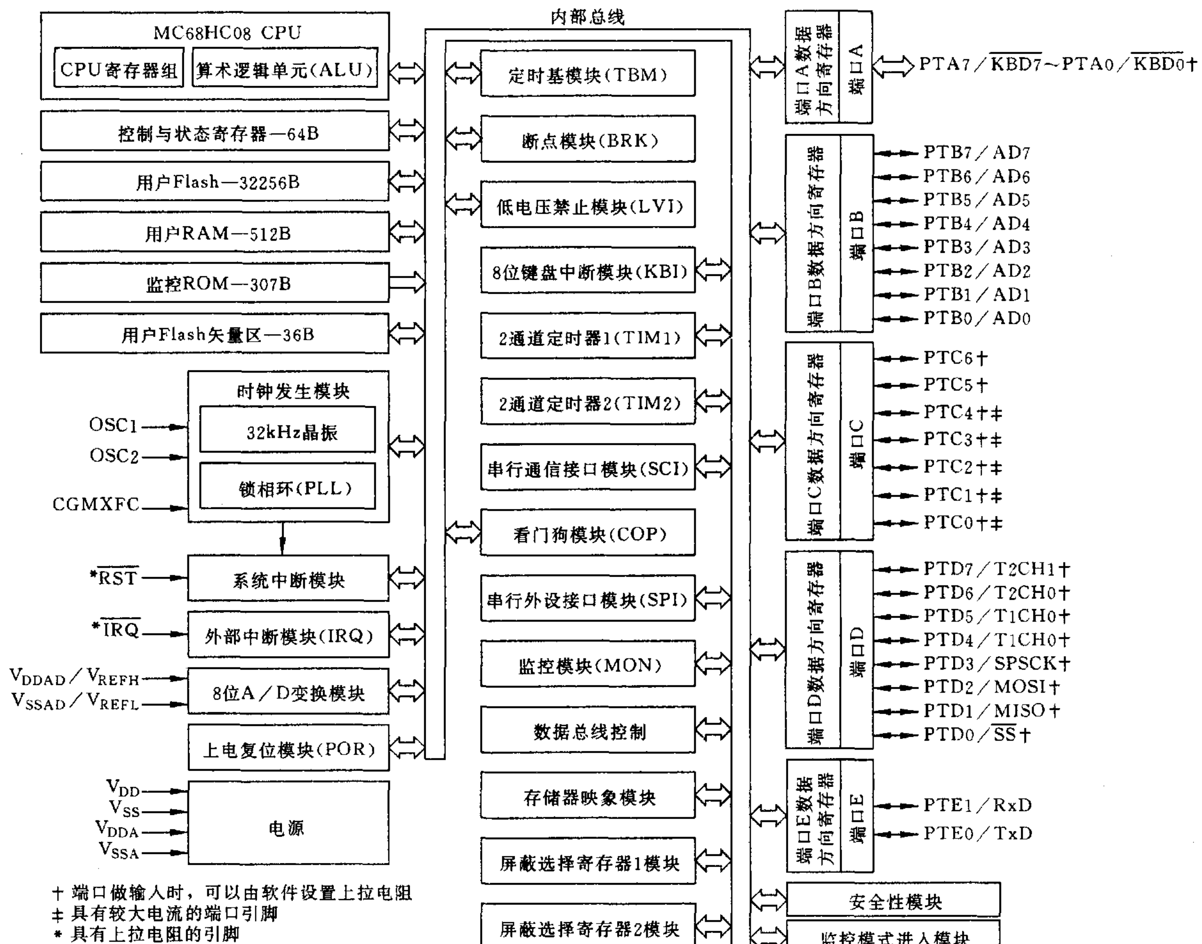


图 2-9　MC68HC908GP32 功能结构框图

Flash 存储器的控制寄存器中还有一个块保护寄存器 FLBPR（$FF7E)，它指出被保护区的首地址，而末地址一律为 $FFFF。被保护区是只读区，不能对它进行擦写操作。FLBR 保护值为

FLBPR= $00，保护全部 Flash 存储器；

FLBPR= $01，保护区为 $8080～ $FFFF；

FLBPR= $02，保护区为 $8100～ $FFFF；

FLBPR= $FE，保护区为 $FF00～ $FFFF；

FLBPR= $FF，不保护。

监控 ROM 在微控制器出厂前已固化在其内部，其中包含了有关系统检测、Flash 编程以及串行通信等功能的代码。这就使得微控制器多了一种不同于正常用户方式的特殊操作方式，称为监控方式。在特定条件下，微控制器可以不进入用户方式，而是进入监控方式。监控 ROM 可以通过单一的一条信号线与主机进行串行通信，接收和执行预先定义的主机命令，如读写存储器、执行程序等，并返回结果。适当运用监控方式和这些主机命令，能够完成一些特殊功能，例如：

(1) 下装代码到 RAM 或 Flash 存储器中；

(2) 执行 RAM 或 Flash 中的程序代码；

(3) Flash 存储器的加密；

(4) Flash 存储器擦除、写入、校验；

(5) 与主计算机进行标准的不归零传号/空号串行通信，波特率为 4.8k～28.8kbaud；

(6) 在线编程；

(7) 用户方式 Flash 编程。

3. 定时器接口模块

微控制器中的定时器用于定时操作及与定时器有关的 I/O 操作。

MC68HC908GP32 的定时器接口模块（timer interface module, TIM）较之 MC68HC05 有较大改进。MC68HC908GP32 有两个定时器接口模块 TIM1 和 TIM2，具有定时器溢出、输入捕捉、输出比较和脉宽调制功能。每个 TIM 有以下特点：

(1) 两个输入捕获/输出比较通道；

(2) 缓冲或非缓冲脉宽调制；

(3) TIM 时钟可编程为内部总线时钟的 7 种分频值；

(4) 自由运行或取模加 1 计数操作；

(5) 溢出时变换通道；

(6) 计数器可停止或复位。

4. 定时基模块

定时基模块(time base module，TBM)产生周期性中断，可选择 8 种速率。它由定时基模块控制寄存器 TBCR（$001C）的 TBR2 ～ TBR0 位所控制，在晶振频率为

32.768kHz 时，TBM 产生的中断速率如表 2-1 所示。

表 2-1 定时基速率选择(晶振频率为 32.768kHz)

TBR2	TBR1	TBR0	分频	时基中断速率	
				Hz	ms
0	0	0	32,768	1	1000
0	0	1	8192	4	250
0	1	0	2048	16	62.5
0	1	1	128	256	约 3.9
1	0	0	64	512	约 2
1	0	1	32	1024	约 1
1	1	0	16	2048	约 0.5
1	1	1	8	4096	约 0.24

5. 系统操作正常监视模块

系统操作正常监视模块(computer operating properly,COP)俗称看门狗电路，其功能是在微控制器工作不正常时，产生一个复位信号。该模块有一个计数器，COP 允许后，软件必须周期性地向 $FFFF(COP 控制寄存器)写入任意值，以清除 COP 计数器。若系统由于某种原因使软件工作不正常时，COP 计数器就得不到清零。那么当它溢出时便产生复位信号，以防止程序进入不可预料的操作。在系统设置寄存器中可以设置 COP 速率及允许、禁止 COP。

6. 并行 I/O 接口

并行 I/O 接口是 I/O 接口中最常用的。MC68HC08 有多个并行口，其中有 8 位口、7 位口，也有 2 位口。MC68HC908GP32 的并行 I/O 接口有 5 个口。

(1) PA 口

PA 为双向 I/O 口，作输入时可具有上拉电阻。在允许时，PA 可用作键盘中断输入。键盘中断的触发可以选择为下跳变有效或下跳变和负电平有效。

(2) PB 口

PB 为双向 I/O 口，也可用作 A/D 输入(这时不受 DDRB 所控制)。

(3) PC 口

PC 为 7 位双向 I/O 口，在作输入时，可具有上拉电阻。

(4) PD 口

PD 为 8 位双向 I/O 口，它也用作定时器和 SPI 引脚。在作输入时，可具有上拉电阻。

(5) PE 口

PE 为 2 位双向 I/O 口，它也用作 SCI 引脚(这时它不受 DDRE 影响)。

7. 异步串行通信接口模块

异步串行通信接口模块(serial communication interface，SCI)的用途是能实现诸如 RS-232、RS-485 等类能使用异步串行通信规程的通信，最主要的是用于和其他计算机的数据传输。SCI 的主要功能是：

(1) 全双工高速非归零通信；

(2) 独立式发送和接收操作；

(3) 可编程波特率；

(4) 硬件奇偶校验(MC68HC05 无此功能)；

(5) 噪声检测(MC68HC05 无此功能)。

MC68HC908GP32 的 SCI 比 MC68HC05 的 SCI 功能强，它具有硬件奇偶校验、噪声检测等功能。它有 3 个控制寄存器(SCC1、SCC2、SCC3)和 2 个状态寄存器(SCS1、SCS2)。

8. 串行外设接口

具有主从工作方式的全双工同步串行外设接口(serial peripheral interface，SPI)用于同步串行通信，也可以用于扩展并行接口、存储器、LCD 驱动电路等。

MC68HC908GP32 的 SPI 功能与 MC68HC05 的 SPI 功能基本相同。但前者比后者增加 2 个出错标志，有分开的接收与发送中断，并有灵活的 I/O 脚控制。MC68HC08GP32 的 SPI 有 2 个控制和状态寄存器。

9. 断点模块

断点模块(break module，BRK)可以在设定的地址处产生一个中断，该中断称为断点中断，它使 CPU 中止当前程序的执行而进入中断服务程序。

断点中断可由下述 2 种方式引起。

(1) CPU 产生的地址(该地址在程序计数器中)与断点地址寄存器的内容相匹配时产生断点中断。

(2) 用软件向断点状态与控制寄存器的 BRKA 位写 1 时产生断点中断。

当这 2 种情况之一发生时，断点模块就产生一个断点信号(BKPT)，使 CPU 在结束当前指令后，将一条 SWI 指令装入内部指令寄存器作为下一条指令执行。这样就如同发生一个软件中断，$FFFC 和 $FFFD(在监控模式下为 $FEFC 和 $FEFD)指定了中断服务例程的起始地址。在断点服务例程中执行 RTI 指令，就结束了断点中断，使微控制器恢复到正常的程序流程。

10. A/D 转换器

MC68HC908GP32 具有 8 路 8 位 A/D 转换器，它有一个状态和控制寄存器 ADSCR ($003C)。

11. 存储器直接存取模块

存储器直接存取(direct memory access,DMA)是一种高速的数据传输方式,它可实现存储器与存储器、存储器与外设之间数据的直接传送。有些型号的 MC68HC08,如 MC68HC08XL36 有 3 路 DMA 通道。

12. 模糊控制模块

MC68HC08 的 KX、KJ 系列,含有模糊控制模块,通过几条指令就可实现对被控制对象的模糊控制。模糊控制是近年来越来越引起重视,并已用于军事、控制、家电等领域中的一种新型的控制方法。

13. 键盘中断模块

MC68HC908GP32 的键盘中断模块(keyboard interrupt,KBI),通过端口 A(PA)的 8 个引脚提供 8 个独立的可屏蔽的外部中断。它们既可作为键盘中断,又可作为普通的中断源,这就大大增加了外中断的个数。

14. 时钟发生模块及锁相环电路

时钟发生模块包括晶振电路、锁相环电路和基本时钟电路。

时钟发生模块的晶振电路采用由晶振、电容、电阻组成的通用晶振电路,或者采用外部时钟源。

锁相环电路使微控制器在外部使用较低频率(如 32kHz)晶振时在内部却可以得到较高(如 8MHz)的总线频率,这是个非常成功的有效的电路。有关锁相环的工作原理、理论计算及使用方法,将在后面的章节中做详细介绍。

15. 低电压禁止模块

低电压禁止模块(low voltage inhibition,LVI)的作用就是监测加在 V_{DD}上的供电电压。当 V_{DD}低于某个预定电压值 LVI_{TRIP}时,认为发生电源故障,产生中断信号并强制系统复位。

16. 复位与中断模块

MC68HC908GP32 具有上电复位(POR)、计算机工作正常(COP)、低电压禁止复位、非法指令码和非法地址等复位源。MC68HC908GP32 共有 24 个中断源。

17. 其他模块

MC68HC08 各专用系列的微控制器会有其特殊的模块,如数字信号处理器模块、电话控制模块等等,今后研制出的新型号还会增添新的模块,在本教材中就不一一做介绍了。

2.2.3 MC68HC908GP32 的存储器组织与空间分配

本节以 MC68HC908GP32 为例说明 MC68HC08 存储器的组织及空间分配。MC68HC908GP32 用 CPU08 作中央处理器，CPU08 的寻址范围是 64kB。

1. 存储器组织

MC68HC08 采用将 I/O 寄存器和各类存储器统一编址的方法来组织存储器空间。这种方法与 I/O 寄存器、存储器分别编址及方法相比，节省了硬件，指令格式也相同，但程序的可读性差一些。

2. 存储器空间分配

MC68HC908GP32 的 64kB 存储空间分为以下 12 个区域，如图 2-10 所示。

(1) $0000～$003F，64 字节片内 I/O 寄存器地址。

该区域包含的 I/O 寄存器主要包括端口为 A、B、C、D、E 的数据寄存器和数据方向寄存器；A、B、C 口输入上拉允许寄存器；同步串行口 SPI 的数据、状态、控制寄存器；异步串行口 SCI 的数据、状态、控制及波特率寄存器；定时器的状态、控制、计数及计数模数寄存器；每个定时器的通道 0 及通道 1 的数据、状态和控制寄存器；模数转换 A/D 的输入时钟数据、状态和控制寄存器；外部中断 IRQ 状态和控制寄存器；键盘中断允许控制寄存器；时钟模块控制寄存器；系统配置寄存器；系统集成模块及锁相环电路的各个有关寄存器等。

(2) $0040～023F，512 字节片内 RAM 区。

MC68HC908GP32 共有 512B RAM 用作随机存储器 RAM 与堆栈。

复位后，堆栈指针为 $00FF，堆栈区位于 RAM 的第 0 页。但是 MC68HC908GP32 的堆栈区的位置是可以编程的，16 位的堆栈指针 SP 使堆栈可以处于 64kB 存储空间中的任意位置。为了正确操作，堆栈指针 SP 必须指向 RAM 区。在中断处理之前，CPU 要使用堆栈的 5B 保存 CPU 寄存器(A、X、PC、CCR)的内容。为了与 MC68HC05 兼容，MC68HC08 的 H 寄存器没有入栈。在子程序调用的时候，CPU 要使用堆栈的 2B 去存放返回地址(PC 值)。入栈操作使堆栈指针 SP 的值减小，出栈操作使堆栈指针的值增加。在进行嵌套调用时，尤其要注意的是不要使堆栈溢出。

存储器的第 0 页是 256B 的 RAM，它分为 64B 的 I/O 区和 192B 的用户 RAM 区。由于堆栈指针是可以编程的，因此当堆栈指针从复位时的 $00FF(第 0 页)移出时，整个第 0 页的 RAM 空间就可以全部用于 I/O 控制和存放用户的数据和代码。这样，那些只适用于第 0 页的直接寻址指令便能快速而有效地存取第 0 页 RAM 空间。因此，第 0 页就成为用户储存那些访问频率较高的全局变量的理想空间。

(3) $0240～$07FF，不存在的存储区。

在存储器分配图和寄存器图中，阴影表示的是不存在的存储器。当存取这些单元时可以引起非法地址复位。编程时要切记这一点，不要使用这些单元。

地址	内容
$0000 ↓ $003F	I/O 寄存器 64B
$0040 ↓ $023F	RAM 512B
$0240 ↓ $7FFF	不存在的单元 32192B
$8000 ↓ $FDFF	Flash 存储器 32256B
$FE00	系统集成模块断点状态寄存器 (SBSR)
$FE01	系统集成模块复位状态寄存器 (SRSR)
$FE02	保留 (SUBAR)
$FE03	系统集成模块断点标志控制寄存器 SBFCR
$FE04	中断状态寄存器 (INT1)
$FE05	中断状态寄存器 (INT2)
$FE06	中断状态寄存器 (INT3)
$FE07	保留
$FE08	Flash控制寄存器 (FLCR)
$FE09	断点地址寄存器高 (BRKH)
$FE0A	断点地址寄存器低 (BRKL)
$FE0B	断点状态和控制寄存器 (BRKSCR)
$FE0C	低电压禁止状态寄存器 (LVISR)
$FE0D $FE0F	不存在的单元 3B
$FE10 ↓ $FE1F	不存在的单元 16B
$FE20 ↓ $FF52	监控 ROM 307B
$FF53 ↓ $FF7D	不存在的单元 43B
$FF7E	Flash 块保护寄存器 (FLBPR)
$FF7F ↓ $FFDB	不存在的单元 93B
$FFDC ↓ $FFFF	Flash 矢量区

图 2-10　MC68HC08 存储器的空间分配

(4) ＄8000～＄FDFF,32256 字节的 Flash 区。

这是用户的程序区。Flash 是一种电可擦可写的存储器,容量大,擦写速度快,而且 MC68HC908GP32 具有在片的电荷泵,可以产生编程和擦除用的高压,所以只需要单一的外部电源就可以实现 Flash 的读出、写入及擦除工作。详细内容参照有关章节。

(5) ＄FE00～＄FE0C,第二段 I/O 寄存器区。

大部分的控制、状态和数据寄存器都在存储器第 0 页范围内(＄0000～＄003F),详见附录,其余的 I/O 寄存器具有它们自己的地址,分配如下:

＄FE00:SBSR,系统集成模块断点状态寄存器;

＄FE01:SRSR,系统集成模块复位状态寄存器;

＄FE02:SUBAR,保留;

＄FE03:SBFCR,系统集成模块断点标志控制寄存器;

＄FE04:INT1,中断状态寄存器 1;

＄FE05:INT2,中断状态寄存器 2;

＄FE06:INT3,中断状态寄存器 3;

＄FE08:FLCR,Flash 控制寄存器;

＄FE09:BRKH,断点地址寄存器高位;

＄FE0A:BRKL,断点地址寄存器低位;

＄FE0B:BRKSCR,断点状态和控制寄存器;

＄FE0C:LVISR,低电压禁止状态寄存器;

＄FF7E:FLBPR,Flash 块保护寄存器。

(6) ＄FE0D～＄FE0F,3 个字节不存在的存储区。

(7) ＄FE10～＄FE1F,16 字节为 A 系列监控代码保留区。

(8) ＄FE20～＄FF52,307 字节监控 ROM。

(9) ＄FF534～＄FF70,48 字节不存在的存储区

(10) ＄FF7E,Flash 块保护寄存器。

(11) ＄FF7F～＄FFDB,93 字节不存在的存储区。

(12) ＄FFDC～＄FFFF,36 字节矢量表。

这一区域也由 Flash 存储器组成,其中＄FFF6～＄FFFD 是预留的 8 个保密字节。

2.2.4 MC68HC908GP32 的系统设置

MC68HC908GP32 有两个系统设置寄存器 CONFIG1(＄001F)和 CONFIG2(＄001E),它们的主要功能为选择 SCI 波特率时钟源(内部总线时钟或外部振荡器);确定晶振在 STOP 状态下是否继续运行;确定在 STOP 方式下低电压禁止模块 LVI 是否有效;选择 LVI 为 5V 或 3V 方式;选择 STOP 的恢复时间;允许或禁止 STOP 指令;允许或禁止 COP 功能;选择 COP 速率;低电压禁止模块的各项控制等。

2.2.5 MC68HC908GP32 的管脚与封装

MC68HC908GP32 有 3 种封装形式:40 脚 PDIP、42 脚 SDIP 及 44 脚 QFP。40 脚 PDIP 如图 2-11 所示,42 脚芯片比 40 脚芯片多了 PTD6 和 PTD7 2 个 I/O 管脚,44 脚芯片比 40 脚芯片多了 PTC5、PTC6 及 PD6、PD7 等 4 个 I/O 管脚,其余信号都是相同的。

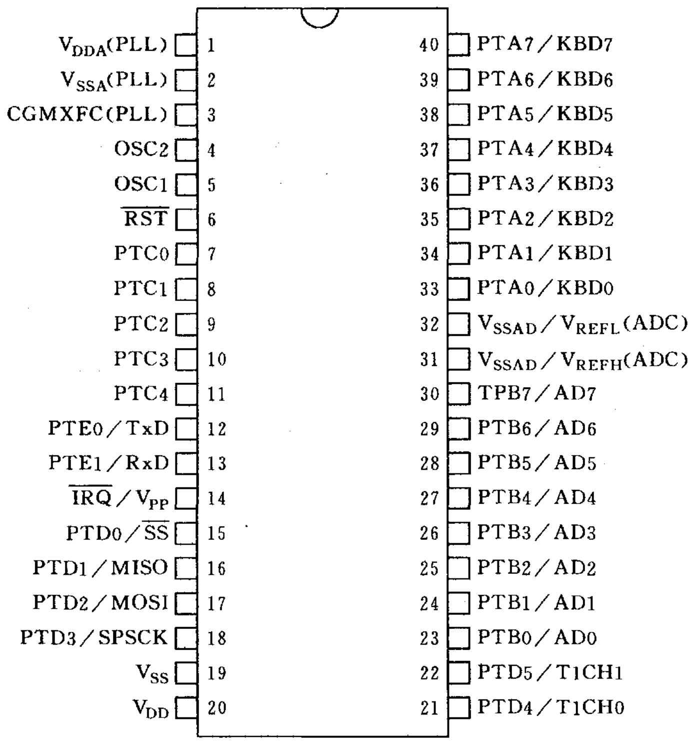


图 2-11 MC68HC908GP32 40 芯 PDIP 封装管脚图

(1) V_{DD}和 V_{SS}:电源供给端。MC68HC908GP32 采用单一电源供电。一般在 V_{DD}与 V_{SS}间接入旁路电容。

(2) OSC1 和 OSC2:振荡器引脚。

(3) $\overline{RST}$:低有效复位输入或输出脚。当任意一个内部中断源有效时,该管脚输出低电平,有内部上拉电阻。

(4) $\overline{IRQ}$:外部中断输入脚,有内部上拉电阻。

(5) V_{DDA}和 V_{SSA}:时钟发生模块模拟部分的电源供给端,与数字电源相似,这两个管脚间也应有去耦电路。

(6) CGMXFC:时钟发生模块的外部滤波电容连接脚。

(7) V_{DDAD}和 V_{SSAD}:A/D 转换器电源供给端,两管脚间也应去耦。

(8) V_{REFH}和 V_{REFL}:A/D 转换器的高和低参考电压输入端。它连至与 V_{DDAD}、V_{SSAD}相同的电位上。

(9) PTA7/KBD7～PTA0/KBD0:8 位通用双向 I/O 口。每一位都可编程为键盘输入脚。作输入时,可选择有上拉电阻。

(10) PTB7/AD7～PTB0/AD0:8 位通用双向 I/O 口,可用作 A/D 输入。

(11) PTC6～PTC0:7 位通用双向 I/O 口。作输入时,可选择有上拉电阻。

(12) PTD7/T2CH1～PTD0/ SS:8 位特殊功能、双向 I/O 口。PTD0～PTD3 可用作 SPI 脚,PTD4～PTD7 可分别用于定时器模块 TIM1 和 TIM2。在作输入时,可选择有上拉电阻。

(13) PTE1/RXD、PTE0/TXD: 2 位通用双向 I/O 口。它们可用作 SCI 脚。

注意: 任何一个没用的输入管脚和 I/O 端口都应当连接到适当的逻辑电平(V_{DD}或V_{SS})上。尽管 MC68HC908GP32 不需要终端负载,但为了减小静态故障的可能性,还是推荐使用终端负载。

第3章　中断与复位

3.1　中断

MC68HC908GP32 的中断功能能够在当前指令执行结束时响应中断。发生中断时微控制器将 CPU 寄存器的值压入堆栈保存，进入中断处理例程。在中断处理例程结束时，将保存的值弹出堆栈，继续执行进入中断以前的程序。

3.1.1　中断的效果

当中断发生时，首先将当前 CPU 寄存器的值压入堆栈。在中断处理例程结束时，RTI 指令将堆栈中保存的程序寄存器的两个字节的值弹出，恢复被打断的正常程序的执行。需要注意的是，为了同 MC68HC05 系列微控制器兼容，MC68HC08 在进入中断时并不保存 H 寄存器的值，需要用户自己将 H 寄存器的值压入堆栈，在返回时再弹出来。当然如果中断处理例程中不使用 H 寄存器，也可以不保存。一般在中断产生时，CPU 会自动关闭中断，设置中断屏蔽位(I 位)来防止其他中断进入，这样其他中断不论优先权是高是低，都不能打断已经执行的中断处理例程。最后 CPU 将用户自己定义的中断向量地址载入程序计数器，开始执行中断服务例程。

3.1.2　中断源

MC68HC908GP32 的中断源如表 3-1 所示。下面介绍其中的 11 个中断源。

表 3-1　MC68HC908GP32 中断源

中断源	标志位	屏蔽位	中断寄存器中的标志位	中断优先权	中断向量地址
复位	无	无	无	0	$FFFE～$FFFF
SWI 指令	无	无	无	0	$FFFC～$FFFD
IRQ 引脚	IRQF	IMASK1	IF1	1	$FFFA～$FFFB
CGM(PLL)	PLLF	PLLIE	IF2	2	$FFF8～$FFF9
TIM1 通道 0	CH0F	CH0IE	IF3	3	$FFF6～$FFF7
TIM1 通道 1	CH1F	CH1IE	IF4	4	$FFF4～$FFF5
TIM1 溢出	TOF	TOIE	IF5	5	$FFF2～$FFF3
TIM2 通道 0	CH0F	CH0IE	IF6	6	$FFF0～$FFF1

续表

中断源	标志位	屏蔽位	中断寄存器中的标志位	中断优先权	中断向量地址
TIM2 通道 1	CH1F	CH1IE	IF7	7	$FFEE～$FFEF
TIM2 溢出	TOF	TOIE	IF8	8	$FFEC～$FFED
SPI 接收满	SPRF	SPRIE	IF9	9	$FFEA～$FFEB
SPI 溢出	OVRF	ERRIE			
SPI 模式错误	MODF	ERRIE			
SPI 发送空	SPTE	SPTIE	IF10	10	$FFE8～$FFE9
SCI 接收越限	OR	ORIE	IF11	11	$FFE6～$FFE7
SCI 噪声疲劳	NF	NEIE			
SCI 结构错误	FE	FEIE			
SCI 奇偶校验错误	PE	PEIE			
SCI 接收满	SCRF	SCRIE	IF12	12	$FFE4～$FFE5
SCI 输入空闲	IDLE	ILIE			
SCI 发送空	SCTE	SCTIE	IF13	13	$FFE2～$FFE3
SCI 发送完成	TC	TCIE			
键盘	KEYF	IMASKK	IF14	14	$FFE0～$FFE1
ADC 转换结束	COCO	AIEN	IF15	15	$FFDE～$FFDF
时基	TBIF	TBIE	IF16	16	$FFDC～$FFDD

1. SWI 指令中断

SWI 指令为软中断指令，它引起一个不可屏蔽的中断。

2. 断点中断

断点中断使得 CPU 在一个软件可编程的断点处执行 SWI 指令。

3. IRQ 引脚外中断

IRQ 引脚上的逻辑零电平触发外中断，中断状态和控制寄存器如图 3-1 所示。

	位	7	6	5	4	3	2	1	0	
$001D	读	—	—	—	—	IRQF	—	IMASK	MODE	INTSCR
	写						ACK			
	复位	0	0	0	0	0	0	0	0	

图 3-1　中断状态和控制寄存器

这个寄存器中各位的作用如下：

IRQF 为中断标志位。1＝中断请求发生；0＝没有中断请求发生。

ACK 为中断请求确认位。写入“1”确认中断。

IMASK 为中断屏蔽位。1＝允许中断请求；0＝禁止中断请求。

MODE 为中断触发方式选择位。1＝下降沿和低电平触发中断；0＝下降沿触发中断。

4. CGM 中断

在 PLL(锁相环电路)每次进入或离开锁相状态时，CGM 对 CPU 产生一个中断请求。当 LOCK 位改变状态时，PLL 标志(PLLF)被置为 1。PLL 中断允许位(PLLIE)允许 PLLF 对 CPU 的中断请求。LOCK 在 PLL 带宽控制寄存器中，PLLF 在 PLL 控制寄存器中。

5. TIM1 中断

(1) TIM1 溢出标志(TOF)。当 TIM1 的计数值达到 TIM1 计数器模式寄存器的值又返回 $0000 时，TOF 位被置为 1。TIM1 溢出中断允许位 TOIE 允许 TIM1 的溢出中断。TOF 和 TOIE 都在 TIM1 的状态和控制寄存器中。

(2) TIM1 通道标志(CH1F～CH0F)。当通道 x 发生输入捕捉或输出比较时，位 CHxF 被置为 1。通道 x 中断允许位 CHxIE 允许通道 x 的 CPU 中断请求。CHxF 和 CHxIE 都在 TIM1 通道 x 的状态和控制寄存器中。

6. TIM2 中断

(1) TIM2 溢出标志(TOF)。当 TIM2 的计数值达到 TIM2 计数器模式寄存器的值又返回 $0000 时，TOF 位被置为 1。TIM2 溢出中断允许位 TOIE 允许 TIM2 的溢出中断。TOF 和 TOIE 都在 TIM2 的状态和控制寄存器中。

(2) TIM2 通道标志(CH1F～CH0F)。当通道 x 发生输入捕捉或输出比较时，CHxF 被置为 1。通道 x 中断允许位 CHxIE 允许通道 x 的 CPU 中断请求。CHxF 和 CHxIE 都在 TIM2 通道 x 的状态和控制寄存器中。

7. SPI 中断

(1) SPI 接收满标志位(SPRF)。每次当一个字节的数据从移位寄存器传送到 SPI 的接收寄存器中时，SPRF 标志位被设置为逻辑“1”。SPI 接收中断允许位 SPRF，允许 SPRF 对 CPU 的中断请求。SPRF 标志位在 SPI 控制和状态寄存器中，SPRIE 标志位在 SPI 控制寄存器中。

(2) SPI 发送空标志位(SPTE)。每次当一个字节的数据从发送寄存器传送到 SPI 的移位寄存器中时，SPTE 标志位被设置为逻辑“1”。SPI 发送空中断允许位 SPTIE 允许 SPTE 对 CPU 的中断请求。SPTE 标志位在 SPI 控制和状态寄存器中，SPTIE 标志位在 SPI 控制寄存器中。

(3) 模式错误标志位(MODF)。当模式错误允许位(MODFEN)设置为逻辑“1”时，即在从 SPI 模式下，如果 SS 引脚在发送中变成逻辑高；在主 SPI 模式下，如果 SS 引脚在任何时候变成逻辑低，MODF 标志位均会变成逻辑“1”。错误中断允许位 ERRIE 允许 MODF 对 CPU 的中断请求。MODF、MODFEN 和 ERRIE 在 SPI 的状态和控制寄存器中。

(4) 溢出标志位(OVRF)。当软件将一个字节移入移位寄存器时，如果上一次接收的字节还没有读取，OVRF 标志位将被设置为逻辑“1”。错误允许标志位 ERRIE 允许 OVRF 对 CPU 的中断请求。OVRF 和 ERRIE 都在 SPI 状态和控制寄存器中。

8. SCI 中断

(1) SCI 发送空标志位(SCTE)。当 SCI 数据寄存器将数据传送到发送移位寄存器时，SCTE 被设置为逻辑“1”。SCI 发送中断允许位 SCTIE 允许发送 CPU 的中断请求。SCTE 在 SCI 的状态寄存器 1 中，SCTIE 在 SCI 控制寄存器 2 中。

(2) 发送完成标志位(TC)。当发送移位寄存器和 SCI 数据寄存器均为空，并且没有中断或者空闲字符产生时，TC 被设置为逻辑“1”。发送完成中断允许位 TCIE 允许发送 CPU 中断请求。TC 在 SCI 的状态寄存器 1 中，TCIE 在 SCI 控制寄存器 2 中。

(3) SCI 接收满标志位(SCRF)。当 SCI 移位寄存器将数据传送到数据寄存器时，SCRF 被设置为逻辑“1”。SCI 接收满中断允许位 SCRIE 允许发送 CPU 的中断请求。SCRF 在 SCI 的状态寄存器 1 中，SCRIE 在 SCI 控制寄存器 2 中。

(4) 空闲输入标志位(IDLE)。当 10 个或者 11 个逻辑“1”被移入 RxD 引脚时，IDLE 被设置为逻辑“1”。空闲中断允许位 ILIE 允许 IDLE 的 CPU 中断请求。IDLE 在 SCI 的状态寄存器 1 中，ILIE 在 SCI 控制寄存器 2 中。

(5) 接收溢出标志位(OR)。当移位寄存器在上次字节还没有被读取时又移入新的数据，OR 标志位被设置为逻辑“1”。接收溢出允许标志位 ORIE 允许 OR 产生 SCI 错误 CPU 中断请求。OR 在 SCI 的状态寄存器 1 中，ORIE 在 SCI 控制寄存器 3 中。

(6) 噪声疲劳标志位(NF)。当 SCI 在输入的字节，中断字符，包括开始位，数据，停止位中监测到噪声时，NF 被设置为逻辑“1”。噪声错误中断允许位 NEIE 允许 NF 产生 SCI 错误 CPU 中断请求。NF 在 SCI 的状态寄存器 1 中，NEIE 在 SCI 控制寄存器 3 中。

(7) 结构错误标志位(FE)。当 SCI 在停止位处接收到逻辑“0”时，FE 被设置为逻辑“1”。结构错误中断允许标志位 FEIE 允许 FE 产生 SCI 错误 CPU 中断请求。FE 在 SCI 的状态寄存器 1 中，FEIE 在 SCI 控制寄存器 3 中。

(8) 奇偶校验错误标志位(PE)。当 SCI 在接收的数据中监测到奇偶错误时，PE 被设置为逻辑“1”。奇偶校验错误中断允许标志位 PEIE 允许 PE 产生 SCI 错误 CPU 中断请求。PFE 在 SCI 的状态寄存器 1 中，PEIE 在 SCI 控制寄存器 3 中。

9. KBD0～KBD7 键盘中断

在键盘中断引脚上的逻辑“0”触发中断请求。键盘状态和控制寄存器如图 3-2 所示，键盘中断允许控制寄存器如图 3-3 所示。

	位	7	6	5	4	3	2	1	0	
$001A	读	0	0	0	0	KEYF	0	IMASK	MODEK	INTKBSCR
	写	—	—	—	—		ACKK			
	复位	0	0	0	0	0	0	0	0	

图 3-2　键盘状态和控制寄存器

图 3-2 中 KEYF 是键盘中断标志位，为"1"时说明键盘中断正在进行。IMASKK 是键盘中断屏蔽位。MODEK 选择键盘中断触发方式。ACKK 是键盘中断确认位，向 ACKK 写"1"会清除键盘中断请求。

	位	7	6	5	4	3	2	1	0	
$001B		KBIE7	KBIE6	KBIE5	KBIE4	KBIE3	KBIE2	KBIE1	KBIE0	INTKBIER
	复位	0	0	0	0	0	0	0	0	

图 3-3　键盘中断允许控制寄存器

图 3-3 中 KBIE7～KBIE0 是键盘中断允许位，为"1"时允许键盘中断。

10. ADC 中断

当 AIEN 标志位被设置为逻辑"1"时，在每次模数转换结束时，ADC(模数转换器)模块会产生一个 CPU 中断。当中断被允许时，COCO/IDMAS 位不再用作转换结束标志位。

11. TBM 中断

TBM 能够以 TBR2～TBR0 定义的速度产生 CPU 中断。当时基计数器溢出时，TBIF 标志位被设置为逻辑"1"。

3.1.3　中断状态寄存器

1. 中断状态寄存器 1

中断状态寄存器 1 如图 3-4 所示。

	位	7	6	5	4	3	2	1	0	
$FE04	读	IF6	IF5	IF4	IF3	IF2	IF1	0	0	INT1
	写	R	R	R	R	R	R	R	R	
	复位	0	0	0	0	0	0	0	0	

图 3-4　中断状态寄存器 1

IF6～IF1 为中断标志 6～1。

这些标志位代表了表 3-1 中所述的中断源。

1＝中断请求发生；
0＝没有中断请求发生；
位 1 和位 0 总是读为 0；
“R”代表保留为将来使用，以下同。

2. 中断状态寄存器 2

中断状态寄存器 2 如图 3-5 所示。

		7	6	5	4	3	2	1	0	
$FE05	读	IF14	IF13	IF12	IF11	IF10	IF9	IF8	IF7	INT2
	写	R	R	R	R	R	R	R	R	
	复位	0	0	0	0	0	0	0	0	

图 3-5　中断状态寄存器 2

IF14～IF7 为中断标志 14～7。
这些标志位代表了表 3-1 中所述的中断源。
1＝中断请求发生；
0＝没有中断请求发生；
位 1 和位 0 总是读为 0。

3. 中断状态寄存器 3

中断状态寄存器 3 如图 3-6 所示。

		7	6	5	4	3	2	1	0	
$FE06	读	0	0	0	0	0	0	IF16	IF15	INT3
	写	R	R	R	R	R	R	R	R	
	复位	0	0	0	0	0	0	0	0	

图 3-6　中断状态寄存器 3

IF16～IF15 为中断标志 16～15。
这些标志位代表了表 3-1 中所述的中断源。
1＝中断请求发生；
0＝没有中断请求发生；
位 7～位 2 总是读为 0。

3.2 复位

复位能够迅速使微控制器进入到开始状态，并且从用户定义的存储器地址开始执行程序。

3.2.1 复位的效果

复位能够产生如下效果：

(1) 迅速停止当前正在执行的指令。

(2) 初始化控制和状态位。

(3) 从地址 $FFFE～$FFFF 将用户自己定义的中断向量地址送到程序计数器 PC 中。

(4) 选择 CGMXCLK 时钟为总线时钟除以 4。

3.2.2 外部复位

当逻辑低电平加到 RST 引脚一段时间后会产生外部复位。外部复位将 SIM 复位寄存器中的 PIN 位设置为逻辑“1”。

3.2.3 内部复位

芯片的内部复位源将 RST 引脚拉低 32 个 CGMXCLK 周期来复位外部设备。在释放 RST 引脚以后，微控制器还将被置于复位状态 32 个周期。这些内部复位源包括如下 5 种。

1. 上电复位

上电复位(POR)是由 V_{DD} 引脚上的电压正跳变引起的内部复位。上电复位的过程为：

(1) 使 CPU 和其他模块的时钟信号稳定地延时 4096 个 CGMXCLK 周期。

(2) 使 RST 引脚在振荡器稳定之前保持低电平。

(3) 在振荡器稳定之后 32 个 CGMXCLK 周期内释放 RST 引脚。

(4) 在振荡器稳定之后 64 个 CGMXCLK 周期内 CPU 开始从复位向量执行程序。

(5) 在 SIM 复位状态寄存器中将 POR 位和 LP 位设置为逻辑“1”，并且将所有其他标志位清零。

2. 看门狗

看门狗复位是由看门狗计数器溢出引起的内部复位。看门狗复位将 SIM 复位寄存器中的 COP 位设置为逻辑“1”。向地址为 $FFFF 的看门狗控制寄存器中写入任意的数值将清零看门狗计数器，以防止看门狗复位。

3. 低电压禁止复位

低电压禁止复位是由于电源电压降低到低电压禁止电压时产生的内部复位。

低电压禁止复位的过程为：

(1) 使 CPU 和其他模块的时钟信号稳定地延时 4096 个 CGMXCLK 周期。

(2) 使 RST 引脚在振荡器稳定之前保持低电平。

(3) 在振荡器稳定之后 32 个 CGMXCLK 周期内释放 RST 引脚。

(4) 在振荡器稳定之后 64 个 CGMXCLK 周期内 CPU 开始从复位向量执行程序。

(5) 将 SIM 复位状态寄存器中的 LVI 位设置为逻辑"1"。

4. 非法操作码复位

非法操作码复位是由不在指令集中的操作码引起的内部中断。非法操作码复位将 SIM 复位寄存器中的 ILOP 位设置为逻辑"1"。

5. 非法地址复位

非法地址复位是由于从不在控制地址内的地址获取操作码引起的内部复位。非法地址复位将 SIM 复位寄存器中的 ILAD 位设置为逻辑"1"。

3.2.4 SIM 复位状态寄存器

SIM 复位状态寄存器 SRSR 如图 3-7 所示，它是一个只读寄存器，当读取该寄存器的值时，各个标志位被自动清零。

	位	7	6	5	4	3	2	1	0	
$FE01	读	POR	PIN	COP	ILOP	ILAD	0	LVI	0	SRSR
	写	R	R	R	R	R	R	R	R	
	复位	0	0	0	0	0	0	0	0	

图 3-7　SIM 复位状态寄存器

这个寄存器中各位的作用为

(1) POR，上电复位标志。1＝自从上一次读取 SRSR 以来的上电复位；0＝自从上一次上电复位以来的读取 SRSR。

(2) PIN，内部复位标志。1＝自从上一次读取 SRSR 以来的通过 RST 引脚的外部复位；0＝自从上一次外部复位以来的 POR 或者对 SRSR 的读取。

(3) COP，看门狗复位标志。1＝上一次是由 COP 计数器溢出引起的复位；0＝POR 或者是对 SRSR 的读取。

(4) ILOP，非法操作码复位标志。1＝上一次是非法操作码引起的复位；0＝POR 或者是对 SRSR 的读取。

(5) ILAD，非法地址复位标志。1＝上一次是非法地址引起的复位；0＝POR 或者是对 SRSR 的读取。

(6) LVI，低电压抑制复位标志。1＝上一次是低电压抑制引起的复位；0＝POR 或者是对 SRSR 的读取。

3.3 CONFIG 寄存器

CONFIG 寄存器包括 CONFIG2 和 CONFIG1 两个寄存器，地址分别为 $ 001E 和 $ 001F，如图 3-8 和 3-9 所示。CONFIG 寄存器随时可以被读出来。这两个寄存器允许和禁止以下功能：

(1) STOP 模式的恢复时间(32 个或 4096 个 CGMXCLK 时钟周期)。

(2) 看门狗定时器溢出时间($2^{18}\sim2^4$ 或者 $2^{13}\sim2^4$ 个 CGMXCLK 时钟周期)。

(3) STOP 指令的允许与禁止。

(4) 计算机操作正常模块的允许与禁止。

(5) 低电压禁止模块的控制和电压跳变点的选择。

(6) 允许和禁止 STOP 模式下的振荡器。

CONFIG 寄存器主要用于一些功能的初始化。每次复位以后，CONFIG 寄存器的所有位均被清零，而且 CONFIG 寄存器每次复位以后只能写一次。因为这些功能影响微控制器的正常操作，所以最好是在每次上电复位以后就尽快设置 CONFIG 寄存器。

	位	7	6	5	4	3	2	1	0	
$ 001E	读	0	0	0	0	0	0	OSC STOPENB	SCIBDSRC	CONFIG2
	写	R	R	R	R	R	R			
	复位	0	0	0	0	0	0	0	0	

图 3-8 CONFIG 寄存器 2

		7	6	5	4	3	2	1	0	
$ 001F	读 写	COPRS	LVISTOP	LVIRSTD	LVIPWRD	LVI50R3	SSREC	STOP	COPD	CONFIG1
	复位	0	0	0	0	0	0	0	0	

图 3-9 CONFIG 寄存器 1

CONFIG 寄存器中各位的作用为

(1) OSCSTOPENB 为振荡器 STOP 模式下允许位。

将 OSCSTOPENB 设置为逻辑“1”$_2$ 使得振荡器在 STOP 模式下也正常工作。这一点对于时基模块在 STOP 模式下产生周期性的唤醒非常有用。

1＝在 STOP 模式下允许振荡器继续工作。

0＝在 STOP 模式下禁止振荡器工作。

(2) SCIBDSRC 为 SCI 波特率时钟源控制位。SCIBDSRC 控制用于 SCI 的时钟，这个控制位的设置影响 SCI 操作的频率。

1＝内部总线时钟用作 SCI 的时钟。

0＝外部振荡器时钟用作 SCI 的时钟。

(3) COPRS 为 COP 速度选择位。COPRS 选择 COP 溢出的范围。复位清除 COPRS 位。

1=COP 溢出范围在 $2^{13}\sim2^{4}$ 个 CGMXCLK 时钟周期。

0=COP 溢出范围在 $2^{18}\sim2^{4}$ 个 CGMXCLK 时钟周期。

(4) LVISTOP 为 STOP 模式下 LVI 允许位。如果 LVIPWRD 位被清除，那么设置 LVISTOP 位将在 STOP 模式下允许 LVI。复位时 LVISTOP 被清除。

1=在 STOP 模式下允许 LVI。

0=在 STOP 模式下禁止 LVI。

(5) LVIRSTD 为 LVI 复位禁止位。LVIRSTD 禁止来自 LVI 模块的复位信号。

1=允许 LVI 复位信号。

0=禁止 LVI 复位信号。

(6) LVIPWRD 为 LVI 电源禁止位。LVIPERD 禁止 LVI 模块。

1=禁止 LVI 模块的电源。

0=允许 LVI 模块的电源。

(7) LVI5OR3 为 LVI 5 伏或者 3 伏操作模式选择位。LVI5OR3 选择 LVI 模块的电压操作模式，LVI 的电压模式应该与 V_{DD} 相同。

1=LVI 工作在 5V。

0=LVI 工作在 3V。

(8) SSREC 为快的 STOP 模式恢复选择位。SSREC 允许 CPU 在延时 32 个 CGMXCLK 周期后退出 STOP 模式，否则 CPU 将延时 4096 个 CGMXCLK 周期才退出 STOP 模式。

1=经过 32 个 CGMXCLK 周期退出 STOP 模式。

0=经过 4096 个 CGMXCLK 周期退出 STOP 模式。

(9) STOP 为 STOP 指令允许位。STOP 位允许 STOP 指令。

1=允许 STOP 指令。

0=STOP 指令为非法指令。

(10) COPD 为 COP 禁止位。COPD 禁止 COP 模块。

1=禁止 COP 模块。

0=允许 COP 模块。

3.4 系统集成模块简介

系统集成模块(SIM)与 CPU 一起控制微控制器，是系统级的控制器。所有中断和复位的时钟信号均由 SIM 提供和控制，它的功能主要包括

(1) 总线时钟的产生和控制，包括 STOP、等待、复位、断点的进入和恢复；内部时钟控制。

(2) 用户复位控制，包括上电复位和 COP 溢出。

(3) 中断控制，包括识别时序；仲裁控制时序；中断地址产生。

(4) CPU 允许和禁止时序。

(5) 可扩展到 128 个中断源的模块结构。

SIM和时钟生成器为CPU提供各种时钟信号。时钟生成器产生一个时钟信号输出到SIM来产生系统时钟，这个时钟信号可以来自外部振荡器，也可以来自内部锁相环电路，可分以下3种情况。

(1) 在用户模式下，内部总线的频率可以是晶体振荡器的输出(CGMXCLK)或锁相环电路输出(CGMVCLK)的4分频。

(2) 当上电复位模块或者低电压禁止模块产生复位信号时，CPU内部时钟保持复位状态直到经过4096个CGMCLK时钟。在这期间，RST引脚被SIM设置为低。内部总线也在4096个CGMCLK时钟之后开始工作。

(3) 在WAIT模式下，CPU时钟并不工作，SIM为其他模块提供时钟。

第 4 章 MC68HC08 指令系统及汇编语言程序设计

开发微控制器离不开汇编语言，汇编语言实际上是机器语言，也就是机器指令与代码。只有熟练掌握这些指令，才能有效地用汇编语言编写程序。

MC68HC08 目标码与 MC68HC05 完全向上兼容。

本章首先介绍 MC68HC08 的寻址方式和指令系统，然后简单介绍 MC68HC08 汇编语言程序设计。

4.1 指令分类

按指令功能 MC68HC08 指令可分为以下 8 大类。

(1) 数据传送指令(data movement)；

(2) 算术运算指令(arithmetic)；

(3) 逻辑指令(logic)；

(4) 数据和位操作指令(data and bit manipulation)；

(5) 程序控制指令(program control)；

(6) BCD 数据处理指令(binary coded decimal data handling)；

(7) 循环指令(loopingconstructs)；

(8) 堆栈处理指令(stack manipulation)。

把指令按功能分类的目的在于便于理解与记忆。

4.2 寻址方式

一条指令看成由操作码和操作数两部分组成，如表 4-1 所示，每一种寻址方式(隐含寻址除外)要求在指令中有两个字节，作为有效地址(EA)。在指令执行存储器访问期间，这个值将出现在地址总线上。

表 4-1 指令与其对应的操作码

寻址方式	隐含(INH)	立即(IMM)	直接(DIR)	扩展(EXT)	变址(IX)	变址(IX1)	变址(IX2)
指令	INCX	LDA #$AB	LDA $50	LDA $350	LDA,X	LDA $10,X	LDA $200,X
机器码	5C	A6 AB	B6 50	C6 03 50	F6	E6 10	D6 02 00

MC68HC08 的寻址方式，粗略可分为隐含寻址(INH)、立即寻址(IMM)、直接寻址(DIR)、扩展寻址(EXT)、相对寻址(REL)、无偏移量变址寻址(IX)、8位偏移量变址寻址(IX1)、16位偏移量变址寻址(IX2)、用栈指针的8位偏移量变址寻址(SP1)、用栈指针的16位偏移量变址寻址(SP2)、存储器到存储器寻址方式(4种)，共16种寻址方式。这是它优于其他公司8位微控制器的一个特点。它使程序员可选择最佳的寻址方式来优化其程序，不仅可方便地访问全部的内存空间(可达64kB)中的各种数据表格、代码等，还可以访问每个字节中的各位。由于有了多种寻址方式，一条指令存取操作数可以有不同的方法，每条指令使用不同的寻址方式对应不同的操作码，所以 MC68HC08 的基本指令需288条不同的指令操作码。其与 MC68HC05 的寻址方式比较见表4-2。

表4-2 MC68HC08 与 MC68HC05 寻址方式比较

寻址方式		HC05	HC08
隐含寻址:(无操作数)		有	有
立即寻址		有	有
直接寻址		有	有
扩展寻址		有	有
变址(3种)	无偏移	有	有
	8位偏移	有	有
	16位偏移	有	有
相对寻址		有	有
堆栈寻址(2种)	8位偏移	无	有
	16位偏移	无	有
存储器到存储器寻址(4种)	立即寻址到直接寻址	无	有
	直接寻址到直接寻址	无	有
	自动变址寻址到直接寻址	无	有
	直接寻址到自动变址寻址	无	有
变址预加1		无	有
变址8位偏移、预加1		无	有

下面分别介绍各种寻址方式。其中有效地址是用于从存储器存取或继续执行程序的地址。另外在指令系统中，一些前缀符号具有特殊的意义，表4-3列出了所有符号的含义。

表4-3 特殊符号的含义

前缀符号	含义	前缀符号	含义
无	十进制数	#	立即数
$	十六进制数	EA	有效地址
@	八进制数	<	强迫为直接寻址
%	二进制数	>	强迫为扩展寻址
’	ASCⅡ码字符	*	当前存储器地址

4.2.1 隐含寻址方式

在隐含寻址方式中，指令的所有有关信息均在操作码中。用隐含寻址方式访问累加器、变址寄存器和条件码寄存器，均为单字节指令，比如 DAA，CLRA，DIV 等。指令中只有操作码，没有操作数。指令的意义是明显的，不需要任何操作数。

例如：CLRA ;把累加器 A 清零

4.2.2 立即寻址方式

在立即寻址方式中，实际操作数紧跟在指令码之后，字节数与寄存器大小一致。这类指令含有 2、3 或 4 个(当需前置字节时)字节，紧跟在操作码后面的 # 是说明符，说明后面紧跟的是操作数值而不是操作数地址。立即数限制为 1 或 2 个字节，它取决于指令所使用的寄存器大小。

例如：LDA # $FF ;把十六进制值 $FF 送到累加器 A 中

4.2.3 直接寻址方式

在直接寻址方式中，操作数的有效地址的低字节放在操作码后面的一个字节中，地址的高位字节默认为 $00，不包含在指令中。绝大部分指令都具有直接寻址方式。它节省了指令空间，提高了指令的执行速度。直接寻址方式只能对 $0000～$00FF 内存空间中的操作数进行操作。有时也将直接寻址方式称为零寻址方式。

直接寻址方式指令比效果相当的扩展寻址指令少用了一个字节的程序内存空间。指令的执行时间也减少了一个时钟周期。对于一个大型程序，这样的节省是很可观的。大部分微控制器将 $0000～$00FF 的内存空出，以便设计者放置那些经常需要调用的数据。

例如：LDA $50 ;把地址为 $0050 的单元内容送到累加器 A 中

4.2.4 扩展寻址方式

在扩展寻址方式中，操作数的有效地址为操作码后面的两个字节。因此，大多数扩展地址指令为 3 个字节，其中 1 个字节是操作码，2 个字节是有效地址。

在汇编语言中，在单字节地址前加符号“＜”表示直接寻址。否则，编译时会在该地址字节前加上一个 $00 字节，就成了扩展寻址。

例如：LDA ＜$40 ;表示汇编时用直接寻址

LDA $40 ;表示汇编时将生成两字节操作数 $0040，属扩展寻址

扩展寻址中，操作数占两个字节。操作数表示的是地址，故寻址范围是 64kB。

例如：LDA $0400 ;把地址为 $0400 的单元内容送到累加器 A 中

4.2.5 变址寻址方式

1. 使用变址寄存器的变址寻址方式

所谓变址寻址就是变址寄存器 H:X 的内容加上指令中给定的 8 位或 16 位偏移量作为操作数的地址。变址寻址又可分为无偏移量、8 位偏移量和 16 位偏移量等 3 种变址寻址方式。

(1) 无偏移量变址寻址(IX)

无偏移量变址寻址方式实际上是间接寻址方式。CPU05 的变址寄存器为 8 位，而 CPU08 的变址寄存器扩展到 16 位，允许用户访问 64kB 的内存空间。如果没有一个指令修改 H 寄存器中的内容，H 寄存器中的值将默认为 \$00，这样就确保了和 CPU05 指令的完全兼容。无偏移量变址寻址指令可以移动链表指针或锁定一个经常使用的 RAM 地址或输入输出地址，它们均为单字节指令。

例如：CLR ,X　　　　;把以 H:X 为地址的单元的内容清零

(2) 8 位偏移量变址寻址(IX1)

在 8 位偏移量变址寻址方式中，有效地址是无符号的 16 位 H:X 的内容和操作码后的一个字节无符号整数之和。它可用于从一个有 N 个元素的表中选择第 K 个元素。这时，K 的值放在 H:X 中，而指令中 8 位偏移量实质上是表格的首地址。

例如：CLR \$10,X　　　　;把以 H:X 加上 \$10 为地址的单元的内容清零

(3) 16 位偏移量变址寻址(IX2)

16 位偏移量变址寻址指令为三字节指令，有效地址为无符号的 16 位 H:X 的内容和操作码后的两个字节 16 位无符号整数之和。

例如：STA \$0100,X　　　;把累加器 A 的内容送到以 H:X 加上 \$100 为地址的单元

2. 使用堆栈指针的变址寻址方式

所谓用堆栈指针的变址寻址就是用堆栈指针代替变址寄存器，分为用堆栈指针的 8 位偏移量变址寻址和用堆栈指针的 16 位偏移量变址寻址。由于与堆栈指针相关的指令要求预先访问一个字节，因此所有与堆栈指针相关的指令要比同样的变址指令多用一个时钟周期。

(1) 用堆栈指针的 8 位偏移量变址寻址

用堆栈指针的 8 位偏移量变址寻址指令为三字节指令，功能与 8 位偏移量变址寻址指令相似。用堆栈指针的 8 位偏移量变址寻址可以方便地访问堆栈中的数据，有效地址是无符号 16 位堆栈寄存器的内容和操作码后无符号整数之和。当中断被屏蔽时，这种寻址方式允许堆栈指针寄存器作为第二个变址寄存器使用。

例如：LDA 5,SP　　　　;把以 SP 加上 \$5 为地址的单元内容取到累加器 A 中

(2) 用堆栈指针的 16 位偏移量变址寻址

用堆栈指针的 16 位偏移量变址寻址指令是四字节指令，有效地址是 16 位堆栈指针

寄存器中的无符号内容与操作码后两个字节 16 位无符号整数之和。

例如：LDA ＄100,SP　　;把以 SP 加上＄100 为地址的单元内容取到累加器 A 中

4.2.6 相对寻址方式

相对寻址主要用于相对转移指令。一般来讲，转移指令（不包括位操作数转移指令）由二个字节组成，其中一字节为操作码，一字节为相对偏移量。相对偏移量为有符号二进制数。如果转移条件为真，则把指令的第二字节的 8 位偏移量加到 PC 中，形成有效转移地址。当转移条件为假时，执行转移指令后的指令。相对寻址的范围为－126～＋129 字节。在使用汇编程序时，用户不必要自己计算偏移量，汇编程序会计算偏移量，并进行校验，看它是否在转移范围内。由 4 个新的转移指令(BLT,BGT,BLE,BGE)通过访问标志位 N,Z,V 来确定相关的有符号操作数的值。

BRA ABC　　;无条件转向标号 ABC 处

4.2.7 存储器到存储器的寻址方式

存储器到存储器的寻址方式有 4 种不同的形式。

(1) 立即寻址到直接寻址

(2) 直接寻址到直接寻址

(3) 自动变址寻址到直接寻址

(4) 直接寻址到自动变址寻址

指令格式为：MOV 源地址，目的地址

1. 立即寻址到直接寻址

直接移动立即数寻址为三字节指令，需要 4 个时钟周期，源地址是一个立即数，通常用来初始化变量和寄存器，操作码后紧跟的操作数将被存储到操作码后第二个字节所指的地址中。用这种寻址方式的传送指令不影响累加器的值。

例如：MOV ＃＄AA,＄F0　　;将立即数＄AA 送到地址＄00F0 处

2. 直接寻址到直接寻址

这种寻址方式为三字节指令，需要 5 个时钟周期，通常用于在存储器间直接移动数据。操作码后第一个字节是源操作数的地址，第二个字节是目的操作数的地址。用这种寻址方式的传送指令不影响累加器的值。这种寻址方式排除了累加器对数据传输的影响，将执行时间从 10 个时钟周期减少到 5 个，这种节省使程序可以用于大量存储器间的数据传输。

例如：MOV ＄00,＄F0　　;将地址＄0000 的内容送到地址＄00F0 处

3. 自动变址寻址到直接寻址

这种寻址方式为两字节指令，执行时间为 4 个时钟周期，通常用来在直接寻址页中的数据块传送。用这种寻址方式的指令不影响累加器的值。H:X 中的内容被存放在直接页中，有效地址由操作码的后继字节给出，移出后，H:X 自动加 1。

例如：MOV X+，$18　　　　;将以 H:X 为地址的内容送到地址 $0018 处，然后 H:X 自动加 1

4. 直接寻址到自动变址寻址

这种寻址方式为两字节指令，执行时间为 4 个时钟周期，通常用于用存储器的内容填充直接页中的数据表格。用这种寻址方式的指令不影响累加器的值。移动后，H:X 自动加 1。

例如：MOV $18,X+　　　　;将地址 $0018 处的内容送到以 H:X 为地址的存储器单元中，然后 H:X 自动加 1

4.3 汇编语言程序设计

4.3.1 机器语言和汇编语言

表 4-1 中列出了一些指令的机器码，是用二进制编码和存储的，这样的编码就称为机器指令或机器码(machine code)。用这种机器指令形式所编写的程序就称为机器语言。机器码不易读懂、不好记忆、易出错，这些都给编写程序带来了很大困难。

为了解决上述困难，人们就创造了一种助记符，用来反映指令的功能和主要特征，用它来代替机器指令，便于人们理解和记忆，这是一个很大的进步。另一方面，指令的另一个重要部分是操作数，它在指令中是立即数或是操作数的地址。在机器码中地址是用 16 位二进制数表示的，如 $0000、$1FFF，这在使用时往往很不方便，特别是在程序中必然有不少转移指令，其中的地址部分，就是要转去执行的指令的存放地址。程序中的指令通常是顺序存放的，但是若程序在调试过程中发现有错误，中间要增加或删去一些指令，则修改处以后的指令的地址全部要改动，于是凡是要转移到这个区域的转移指令中的地址部分均要修改。这给编程带来很大麻烦。而在编程时若用名字来表示地址，并且该名字总是代表某一指令的存放地址，因此当这一指令的存放地址改变时，由该名字表示的地址自动变化，而不需要用户进行改变，这就是符号地址或称为标号。

因此，用上述的助记符以及符号地址或标号书写的程序，就称为汇编语言的源程序。但是计算机只认得机器语言，所有用汇编语言写的程序，只有经过汇编程序翻译成机器语言的程序——目标程序，才能执行。这个翻译过程就称为汇编。

4.3.2 汇编语言源程序的格式

用户的汇编程序由一行行汇编语句组成，每行语句可包括以下 4 个部分：标号、操作码、操作数和注释。

例如：LOOP LDA 0，X　　　;X 变址寻址值传送至累加器 A

每一个语句的操作码说明这条语句的执行功能。操作数可以是地址或数据，也可以是空缺。为了使程序便于编写和阅读，可以给一个语句指定一个标号，还可以适当地加一注释，对语句的作用进行说明。

1. 标号

在一行语句的开头部分为标号部分，它可为以下 2 种形式之一。

(1) 标号的第一个字符是星号 * 时，表示该行为注释行，汇编程序忽略注释行。

(2) 可执行语句标号的第一个字符必须是英文字母，后面可以接英文字母、数字、美元符 $ 和下横线符_。通常用代表某种含义的英文缩写来表示，标号由 1 至 15 个字符组成，标号中使用的大写或小写是不相同的。标号通常用来表示转移语句的转向地址或程序的起始地址或数据地址等。

一个标号在标号部分只能出现一次，否则为重复定义错。

除了一些伪指令外，标号被赋予指令或汇编数据的第一字节的程序计数器值。标号可以用冒号结束，但这不是必需的。冒号不是标号的一部分，它只是用于把标号与语句的其他部分隔开。因此，以下两个程序段是等价的。

```
HERE:   DECA
        BNE     HERE
HERE    DECA
        BNE     HERE
```

一行语句可只包含标号部分，这时汇编程序把当前程序计数器的值赋给该标号。

2. 操作码

操作码部分位于标号后，两者之间至少用一个空格或制表符(TAB)相分隔。操作码部分应包含正确的汇编指令助记符或汇编伪指令。在这个部分中，大写字母将在内部转换成小写字母进行处理。因此，nop、NOP、NoP 为相同的助记符。操作码可分为两类：

(1) 指令码，它们与机器指令一一对应。指令码包括与指令有关的寄存器名。寄存器名不应与指令码用空格隔开。这样，CLRA 意味着清零累加器 A，而 CLR A 意味着清零由符号 A 定义的存储器单元。

(2) 伪指令码，它们由汇编程序所使用，用于控制汇编操作。

3. 操作数

操作数部分的处理方式与操作码有关。如有操作数，它必须在操作码后，并且必须用

一个以上空格或制表符分隔。操作数部分可由符号、表达式组成，它的各个组成部分之间必须用逗号分隔。

操作数部分既指出了指令的寻址方式，又指出了指令的操作数。下面介绍操作数部分的各个组成部分的格式(注：下文中圆括号()表示任选项，尖括号＜ ＞表示一个组成部分)。

(1) MC68HC08 操作数语法

MC68HC08 操作数语法如表 4-4 所示。

表 4-4　MC68HC08 操作数语法

操作数格式	MC68HC08 寻址方式
无操作数	累加器和隐含
＜表达式＞	直接、扩展或相对
#＜表达式＞	立即数
＜表达式＞,X	变址
＜表达式＞,＜表达式＞	位置位或清零
＜表达式＞,＜表达式＞,＜表达式＞	位测试和转移

注意：对于基地址为零的变址指令，即操作数为"0,X"时允许使用",X"的形式。它实际上为间接寻址方式。

(2) 表达式

表达式由符号、常数、运算符和括号组成。表达式的运算结果为操作数。

表达式可由运算符+、—、*、/、%、&、|、^等连接符号和常数或字符 *(表示当前程序计数器的值)组成。

(3) 运算符

允许使用的运算符(与 C 语言相同)及其含义如表 4-5 所示。

表 4-5　MC68HC08 允许使用的运算符及其含义

符号	涵义	符号	涵义
+	加	%	取余数
—	减	&	逻辑与
*	乘	|	逻辑或
/	除	^	异　或

表达式从左到右计值，不使用括号。

(4) 符号

每一个符号对应于一个 16 位整数值。在计算表达式时，用该整数来代替符号。星号 * 用来表示当前程序计数器的值。

(5) 常数

常数为在执行程序过程中不改变的数据。汇编时，常数可采用 5 种格式：十进制、十

六进制、二进制、八进制或 ASCII 码。汇编程序把所有常数变换成二进制机器码，并在汇编列表时以十六进制格式显示。

4. 注释

汇编源程序语句的最后一个部分为注释部分，它应由一个以上的空格或制表符与操作数或操作码(无操作数时)分开。它可包含任何可打印的 ASCII 字符。对于初学者，建议给每条语句写注释。

4.3.3 汇编伪指令

在汇编语言中有一类不与任何处理器指令相对应的伪操作命令——伪指令。汇编伪指令是提供给汇编程序的指令，伪指令没有对应的机器码。

1. ORG——定位伪指令

格式： ORG <表达式> (注释)

例如： ORG $8000

ORG 伪指令把由表达式决定的值($8000)赋给程序计数器。以后的指令汇编后应放在从$8000 开始的存储单元中。如果一个源程序中没有 ORG 伪指令，程序计数器初始值为$0000，即从 0 号单元开始汇编。

2. EQU——赋值伪指令

格式： 标号 EQU <表达式> (注释)

例如： REGBAS EQU $8100

EQU 伪指令把表达式的值赋给前面的标号，因而它能把一个非程序计数器的值赋给前面的标号。该标号不能在程序的其他地方再定义。表达式中不能使用后面定义或没有定义的符号，否则将出错。

3. FCB——字节常数定义伪指令

格式：(标号) FCB 表达式(，表达式，…，表达式)(注释)

FCB 伪指令可以有一个或多个由逗号分隔的操作数。每个操作数的值截取低 8 位，放于目标程序的一个字节中。操作数可以是数字常数、字符常数、符号或表达式。如有多个操作数，则其中一个或多个可以为空(两个相连的逗号)，这时操作数的值置为 0。如果操作数的高 8 位不为 0，则出错。

```
例如：COSA    EQU    $0100
                FCB    $E1,$50,    ,$01
                FCB    $81,$82,$83,$41
```

表示从$0100 单元开始存放$E1、$50、$00、$01、$81、$82、$83、$41。

4. FCC——字符串常数定义伪指令

格式：(标号)　FCC　分隔符　字符串　分隔符

FCC 伪指令用于把一个 ASCII 码字符串存入相连的存储器中。它的第一个字节从现行程序计数器开始存放。标号被赋予字符串中第一个字节的地址。字符串中可有任何可打印的 ASCII 字符。字符串由两个相同的字符开头和结束，它们称为分隔符，可以是任何非空格(或制表符)的可打印字符。在 FCC 后的第一个非空字符为分隔符。

例如：LAB　FCC　'I LOVE MC68HC08'

表示把 ASCII 字符串 I LOVE MC68HC08 放于从地址 LAB 开始的存储单元中。

5. FDB——双字节常数定义伪指令

格式：(标号)　FDB　表达式(，表达式，…，表达式)(注释)

FDB 伪指令可以有一个或多个由逗号分隔的操作数。与每一个操作数相关的 16 位值存入目标程序的两个连续字节中，高位字节在前，低位字节在后(这一点与 Intel 系列的微控制器存放顺序正好相反)。存储地址从现行程序计数器开始。标号被赋予第一个操作数的首地址值。操作数可以是数字常数、符号或表达式。如有多个操作数，则其中的一个或几个可以为空(两个相连的逗号)，这时操作数为两个字节的零。

例如：TABC　FDB　$2021，$0101

表示将值$2021、$0101 存入标号为 TABC 开始的存储单元中。

6. BSZ(或 ZMB)——清 0 存储器伪指令

格式：(<标号>)　BSZ <表达式> (<注释>)

或

(<标号>)　ZMB <表达式> (<注释>)

BSZ(或 ZMB)留出一块存储区，每个字节赋予初始值零。存储区的字节数由表达式的值决定。如表达式中包含了后面定义的或没有定义的符号，或表达式的值为零，则将报错。

例如：LAB1 ZMB 100

表示将 LAB1 开始的 100 个存储单元清零。

7. FILL——存储器置数伪指令

格式：(<标号>) FILL <表达式> ，<表达式>

FILL 伪指令把一块存储区置入一个常数。第一个表达式定义存入存储器的字节常数，它的值应在 0～255 之间。第二个表达式表示置值的存储区的字节数。表达式中不能包含后面定义的或没有定义的符号。

例如：LAB2 FILL　$0F，100

表示用值$0F 初始化从 LAB2 开始的 100 个存储单元。

8. RMB——保留存储器字节伪指令

格式：(<标号>) RMB <表达式>

RMB把程序计数器加上由表达式决定的值，从而保留了一块存储区，它的字节长度等于表达式的值。存储器不进行任何初始化。表达式中不能包含任何后面定义的或没有定义的符号。

例如：HPW RMB 2

表示为HPW保留2个字节的存储区。

9. END——表示汇编程序结束

格式：END (<标号>)

END表示本汇编程序到此结束。其中(<标号>)可以是一个语句标号，也可以是一个有效地址，表示程序运行的起始地址，不加标号或地址，编译将默认此地址为0。

4.3.4 汇编语言程序的编译

MC68HC08汇编语言程序的编辑、编译及调试方法见第15章。

4.3.5 S记录

S记录格式是Motorola公司的十六进制目标文件格式。它将程序和数据用可打印的ASCII格式表示，允许用标准的软件工具来检查目标文件，也可以在传送过程中显示其内容。S记录格式还包括出错检验功能，可保证数据传送的正确性。此外，S记录很容易编辑。

1. S记录内容

S记录实际上是由5个部分组成的字符串的集合。它包括记录类型、记录长度、存储器地址、程序/数据及校验和。每个字节包含两个十六进制字符，第一个字符为字节的高四位，第二个字符为低四位。组成一个S记录的5个部分如表4-6所示。

表4-6 S记录组成

类型	记录长度	地址	程序/数据	校验和

S记录字段的内容见表4-7。

2. S记录类型

S记录共定义了8种类型。它们提供了编码、传送和译码的功能。Motorola的装入

表 4-7 S 记录各字段内容

字段	字符数	内　　容
类型	2	记录类型 S0、S1、S9 等
记录长度	2	记录中除类型和记录长度外的字符对的数目
地址	4,6 或 8	表示数据将装入的存储器地址，地址可为 2、3 或 4 个字节(取决于记录类型)
程序/数据	0～2n	为 0～n 个字节可执行程序(目标程序)或数据或描述信息
校验和	2	它为组成记录长度、地址和程序/数据的所有字符之和的反码的低位字节

程序、记录传送控制程序、交叉汇编程序、文件生成和调试程序等均要使用 S 记录格式。简化了的 MC68HC08 交叉汇编只使用 S1 和 S9 两种类型。

S1——包含程序/数据和两字节地址，它表示程序/数据的存放首地址。

S9——用于 S1 记录的结束记录。地址部分应包含程序的执行起始地址的两字节地址，如不定义，则采用输入时第一次遇到的入口地址或默认此地址为 0。这部分无编码/数据字段。

3. S 记录示例

下面是一个典型的 S 记录。

```
S1238000975454545454DE0113BF50A40F97D60113B75181303132333435363738394142436З
S106802044454609
S9030000FC
```

这个模块包含 2 个 S1 程序和数据记录和 1 个 S9 结束记录。

S1——S 记录类型 S1，表示为程序/数据记录，地址为 2 个字节。

23——十六进制 23(十进制 35)表示后面有 35 个字符对，即后面有 35 个字节二进制数，包括 2 个字节地址、32 个字节程序和数据、1 个字节检验和。

8000——两字节地址($8000)，表示后面程序/数据要装入的存储地址。

97...43——32 个字符对是实际的程序/数据的 ASCII 字节。

63——第一个 S1 记录的校验和。

S9——S 记录类型 S9 表示为一个结束记录。

03——十六进制 03，表示后面有 3 个字符对(3 个字节)。

0000——表示两字节地址，为 0。

FC——S9 记录的校验和。

Motorola S 格式记录类型的完整定义如下：

S0——S 格式文件的第一个记录，以 16 进制 ASCII 码的值的形式记录本文件的文件名，首尾包括记录长度和校验码。

S1——地址为 2 字节的代码/数据记录。

S9——S1 类型文件的结束行记录。

S2——地址为 3 字节的代码/数据记录。

S8——S2 类型文件的结束行记录。

S3——地址为 4 字节的代码/数据记录。

S7——S3 类型文件的结束行记录。

S5——如果有的话，标记本文件总共有多少个 S1、S2 或 S3 记录。

4.3.6 汇编语言程序设计和举例

1. 运算类

(1) 加减法

微机中一般均采用补码来表示有符号的数。对于补码表示的数执行加减运算非常方便，只要直接使用 MC68HC08 的加法和减法指令即可。下面以双字节补码为例，说明编程方法。对于多字节运算，只需同样地再增加几条指令即可。

例 4.1 双字节补码加法子程序 DADD。

功能：($50、$51)+($52、$53)结果存于 $54、$55 中。

程序：

```
DAED:   LDA   $51
        ADD   $53      ;低位字节相加
        STA   $55
        LDA   $50
        ADC   $52      ;高位字节相加
        STA   $54
        RTS
```

说明：本子程序完成把 $50、$51 和 $52、$53 中的两个双字节二进制补码数相加，结果送到 $54、$55 中。数据均为高位在前、低位在后。出口时，不改变两个操作数的内容。如要完成把其他存储单元的内容相加，可修改程序中的 $50、$51 等地址。注意，在第 15 章中介绍的 MC68HC908GP32 在线开发系统中 $50 和 $51 单元已被监控程序占用。

例 4.2 双字节补码减法子程序 DSUB。

功能：($50、$51)－($52、$53)结果存于 $54、$55 中。

程序：

```
DSUB:   LDA   $51
        SUB   $53      ;低位字节相减
        STA   $55
        LDA   $50
        SBC   $52      ;高位字节相减
        STA   $54
        RTS
```

说明:本子程序完成把＄50、＄51 中的双字节补码数减去＄52、＄53 中的数,结果送到＄54、＄55 中。

例 4.3 双字节求补子程序 DNEG。

功能:—(＄50、＄51)结果存于＄50、＄51 中。

程序:

```
DNEG:    NEG    $51      ;低位字节取负
         CLRA
         SBC    $50      ;高位字节取负
         STA    $50
         RTS
```

说明:求补运算为求一个负的原码数(不计符号位)的补码,也可用于求一个补码数的负数。它的运算为求反加 1,也可采用 0 减去该数(即求负)。本例中,采用了后一种方法,先执行低位字节的取负,再执行 0 带借位减去高位字节。这样可节省指令,加快执行速度。为进行比较,下面列出用求反加 1 方法的求补子程序。

```
DNEG1:   COM    $50      ;取反高位字节
         LDA    $51
         COMA            ;取反低位字节
         ADC    #1       ;双字节加 1
         STA    $51
         CLRA
         ADC    $50
         STA    $50
         RTS
```

注意:在执行双字节求反加 1 时,由于 INC 指令不改变进位 C,所以只能用 ADC #1 来完成加 1 运算。由此可见,在使用 MC68HC08 指令时,一定要注意它们对各个标志位的影响,特别是 INC、DEC、NEG、COM 等指令。

(2) 乘法

8 位无符号数乘法可以直接用指令 MUL 实现。

多位无符号数乘法是以 8 位无符号数乘法为基础,通过编程实现的,编程思路见图 4-1。

例 4.4 16 位无符号数乘法

从图 4-1 看出,其方法是分步相乘,部分积相加。此处需注意,在部分积相加时,要考虑进位位。此程序较长但不复杂,此处不再列出。

2. 数制转换

计算机技术中用到各种各样的进制及编码:计算机使用二进制,人写程序用十六进制,输入/输出用人们习惯的十进制,而打印、显示及键盘输入又都是 ASCII 码。因此,这些进制及编码之间的转换是经常碰到的事。其算法、编程方法已在不少汇编语言程序设

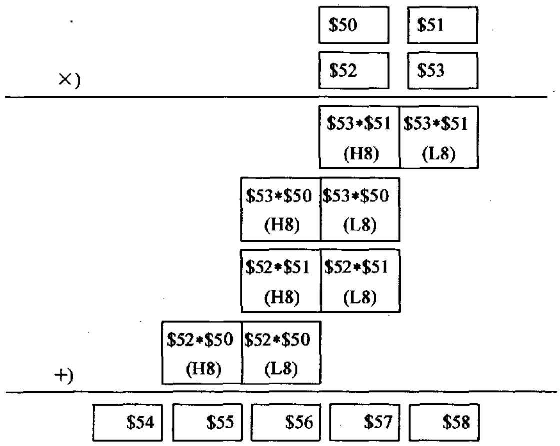


图 4-1　16 位无符号数乘法

计类书籍及微机原理课中作过介绍及训练，此处不再赘述。

3. 散转

散转就是按输入数据的不同转移到不同程序的过程。一般可以采用 JMP 指令或查表法实现。

例 4.5　根据 X 寄存器的值转至相应处理程序。

```
        LSLX                ;X←X*2
        JMP       TAB,X
              ......
TAB:    BRA       PRO0
        BRA       PRO1
        BRA       PRO2
        BRA       PRO3
PRO0:         ......
PRO1:         ......
PRO2:         ......
PRO3:         ......
```

根据键入字符，转至相应程序是实际应用中经常采用的控制方法。通过查表即可转入与键入字符相应的程序，见例 4.6。

例 4.6　根据键入字符 A、B、C、D 转至相应的程序 PROA、PROB、PROC、PROD。假设键入字符的 ASCII 码已存入累加器 A 中。

```
        CLRX
LTB1:   INCX                ;从 TABX+1 开始查表
        CMP     TABX,X      ;比较 A 是否等于表格中的元素
        BNE     LTB2        ;是否相等?
        LSLX                ;相等，把 X 乘以 2
```

```
        JMP     TABY-2,X        ;按 X 散转
LTB2:   CPX     TABX            ;不相等,X 是否等于表长
        BNE     LTB1
                ……
                ……
TABX:   FCB     4
        FCC     'ABCD'
TABY:   BRA     PROA
        BRA     PROB
        BRA     PROC
        BRA     PROD
PROA    ……
PROB    ……
PROC    ……
PROD    ……
```

例 4.7 编程时,若使用一些技巧,则可简化程序的设计。如下两段程序功能相同,但程序长度不同。

```
            ORG     $8000
TABLE1      RMB     4
TABLE2      RMB     32
            ORG     $8100
            LDA     #32
            CLRX
CLRT1:      CLR     $8004,X
            INX
            DECA
            BNE     CLRT1
            END
```

说明:这是表 TABLE2 初始化清零的一段小程序,ORG \$8000 说明表 TABLE1 的起始地址为 \$8000,RMB 4 为表 TABLE1 保留 4 个字节空间,RMB 32 为表 TABLE2 保留 32 字节空间。

```
            ORG     $8100
LEN:        EQU     32
TABLE1:     RMB     4
TABLE2:     BSZ     LEN
```

说明:汇编时,符号 LEN 赋值 32,TABLE1 地址为 \$8100,TABLE2 地址为 \$8104,BSZ 伪指令将自 TABLE2 起始连续 32 字节单元清零,等同于上段程序的功能。

例 4.8 红外遥控信号译码。

在各种各样的家用电器中，大量采用了红外遥控的方式，其原理是将传输的信息用一系列不同宽度（或间隔）的红外脉冲来表示，而接收端通过判断接收到脉冲的宽度（或间隔），从而译出相应的信息。

这里介绍一个应用的实例。这个实例中，红外信号经过前置电路解调后，波形如图4-2 所示。

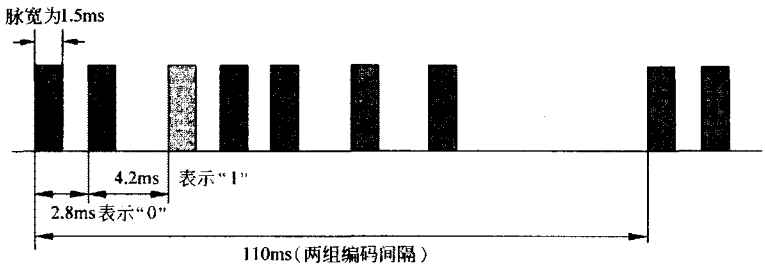


图 4-2　解调后的红外脉冲信号

其中，相邻脉冲的间隔表示了所传递的信息，解码时有两种方案：

(1) 利用微控制器的输入捕捉功能，记录各上跳沿发生的时间，得到各脉冲的时间间隔，从而解出红外遥控信号的指令码。

(2) 利用微控制器的外部中断和定时器，将输入信号接到微控制器的 IRQ(外部中断)端，在中断服务程序中，通过读定时器的值确定各个脉冲出现的时间，减去上一次脉冲出现的时间，得到脉冲间隔，从而解出指令码。

考虑到有些微控制器没有输入捕捉功能，为通用起见，这里以后者为例，讲述软件的具体实现(本例所用微控制器为 MC68HC908GP32)。

程序设计的思路是：用软件模拟一个移位寄存器(RED_DATA)，收到第一个脉冲时，将移位寄存器的值设为 1，以后每收到一个脉冲，便将移位寄存器左移一位，并将计数器加一。如果红外指令收全了，就调用用户子程序，同时将移位寄存器(RED_DATA)清零，以准备下一次接收。

```
*****************************************************************************
** RED_CAPTURE 是红外信号捕捉的主程序
*****************************************************************************
;PULSE_NUMBER    记录接收到的遥控器信号脉冲的个数
;
;
;主程序在初始化阶段，应该先调用下面程序初始化红外遥控接收程序中使用的变量。
INITIALIZE_RED:
MOV    #%00010000,T2SC   ;允许 TIMER2，并且 TIMER2 的时钟源使用内部总线时钟。如
                          果使用 4MHz 的晶体振荡器，则内部总线时钟为 1MHz。因此，
                          2.1ms 的 16 进制为 $0834，1.9ms 的 16 进制为 $076C，1.1ms
                          的 16 进制为 $044C，0.9ms 的 16 进制为 $0384
```

```
CLR     PULSE_NUMBER
CLR     RED_DATA
RTS
;这是外中断子程序：
RED_CAPTURE    EQU    *
    SEI
    PSHH
    TST     PULSE_NUMBER        ；如果 PULSE_NUMBER＝0，说明接收到的是第一个脉冲
                                  信号
    BNE     RED_NEXT            ；否则，不是第一个脉冲信号
    MOV     #%00010000,T2SC     ；使用定时器 2 来测量每两个下降沿之间的时间差，从而确
                                  定接收到的是逻辑“1”还是逻辑“0”
    INC     PULSE_NUMBER        ；PULSE_NUMBER 寄存器保存累计接受到的有效红外
                                  遥控信号的“位”数
    JMP     RED_EXIT
RED_NEXT:
    LSL     RED_DATA
    LDHX    T2CNTH              ；测量两个脉冲之间的时间差，决定接收到的是逻辑“1”或
                                  者逻辑“0”
    MOV     #%00010000,T2SC     ；复位 Timer2，准备下一次计数
    CPHX    #$0834              ；判断上次接受到的脉冲间隔是逻辑“1”还是逻辑“0”，如果
                                  时间差大于 2.1ms，说明接受到的不是正确信号，忽略，重
                                  新计数
    BHI     RED_BAD_BIT
    CPHX    #$076C              ；如果时间差在 1.9ms 到 2.1ms 之间，则判定为逻辑“1”
    BHI     RED_SET
    CPHX    #$0384
    BLS     RED_BAD_BIT         ；如果时间差小于 0.9ms，说明接受到的不是正确信号，忽
                                  略，重新计数
    CPHX    #$044C              ；如果时间差在 1.1ms 到 1.9ms 之间，也认为是错误信号，
                                  忽略
    BHI     RED_BAD_BI
    INC     PULSE_NUMBER
    BCLR    0,RED_DATA          ；如果接收到的是逻辑“0”，将 RED_DATA 的最低位设置
                                  位 0
    JMP     RED_OK
RED_SET:
    INC     PULSE_NUMBER        ；如果接收到的是逻辑“1”，将 RED_DATA 的最低位设置
                                  位 1
    BSET    0,RED_DATA
RED_OK:
    LDA     PULSE_NUMBER        ；PULSE_NUMBER 是否为 9，也就是判断是否接收到了 8
```

```
                                        位数据
    CMP     #$09
    BNE     RED_EXIT
    CLR     PULSE_NUMBER        ；如果接收到了8位数据，清除PULSE_NUMBER，准备下
                                        一次接收
    JSR     DEAL_WITH_RED_REMOTE   ；处理接收到的红外遥控数据。用户自己可以
                                          灵活处理
    CLR     RED_DATA
    JMP     RED_EXIT
RED_BAD_CMD:
    MOV     #$01,PULSE_NUMBER   ；将错误的这次脉冲记为第一次脉冲
    CLR     RED_DATA
RED_EXIT:
    PULH
    CLI
    RTS
```

第5章 闪速存储器

5.1 闪速存储器概述

理想的存储器应具有密度高、读写快、价位低和不挥发的特点，而传统的存储器却只能满足这些要求中的一部分。闪速存储器 Flash 的推出，恰好同时实现了所有这些优良的存储器特性。

Flash 是一种高密度、真正不挥发的高性能读写存储器，兼有功耗低、可靠性高等特点。与传统存储器相比，Flash 具有的主要优势在于：

(1) 固有不挥发性：不像 SRAM（静态随机存取存储器），Flash 无需后备电源来保证数据不变；不同于 DRAM（动态随机存取存储器），Flash 不需要使用数据存储技术来为它提供后备存储。

(2) 易更新性：相对于 EPROM（可擦除可编程的只读存储器）的紫外线擦除工艺，Flash 的电擦除功能为开发者节省了时间，也为用户更新存储器内容提供了可能。而与 EEPROM（电可擦可编的只读存储器）相比较，Flash 的成本更低，密度和可靠性更高。

表 5-1 是 Flash 与传统存储器的比较。

表 5-1 Flash 与传统存储器技术的比较

存储器	固有不挥发性	高密度	低功耗	单晶体管单元	在线可重写
Flash	√	√	√	√	√
SRAM					√
DRAM		√			√
EEPROM	√		√		√
OTP/EPROM	√	√	√	√	
掩膜 ROM	√	√	√	√	

根据工艺 Flash 主要有两类：NAND Flash 和 NOR Flash。NOR Flash 是在 EEPROM 的基础上发展起来的，它的存储单元由 N-MOS 构成，而连接 N-MOS 单元的线都是独立的。NOR Flash 的特点是可以随机读取任意单元的内容，适合于程序代码的并行读写存储，所以常用于制作 PC 的 BIOS 存储器和微控制器内部存储器等等。而 NAND Flash 将几个 N-MOS 单元用同一根线连接，可以按顺序读取存储单元的内容，适合于数据或文件的串行读写存储。

在微控制器内部集成 Flash，并不是 Motorola 首创的技术。在 Microchip 的 PIC 系列、Atmel 的 AT89C51 系列等微控制器中早已引入了片内 Flash 技术。与 ROM 或 EPROM 相比，早期这类微控制器的片内 Flash 在可靠性和稳定性上仍存在着许多不足。

Motorola 在 Flash 技术已相当成熟的时候才推出集成片内 Flash 的 8 位微控制器，在应用方便性和可靠性上均有不俗的表现，具体体现在：

(1) 单一电源电压供应。Flash 在正常的只读情况下，只需要用户为其提供普通的工作电压(如+5V)。如果用户要对 Flash 编程，需要同时提供高出正常工作电压的编程电压(如+9V)。Motorola 通过在片内集成电荷泵，只需要用户提供单一+5V 工作电压，便可以在片内产生出编程电压。这使得用户无需为 Flash 的编程而在目标板上增加额外的电源。

(2) 可靠性高。Motorola 片内 Flash 的存储数据可以保持 10 年以上，可擦写次数也在 1 万次以上。

(3) 擦写速度快。Motorola 片内 Flash 的整体擦除时间可以控制在 5ms 以内，对单字节的编程时间也在 40μs 以内。

最重要的是，Motorola 片内 Flash 支持在线编程，允许微控制器内部运行的程序去改写 Flash 存储内容。这一技术大大增加了 Motorola 微控制器的应用范围和使用方便性。

5.2 闪速存储器的编程操作

5.2.1 闪速存储器编程操作概述

Motorola 片内 Flash 的编程操作分为：擦除和写入。擦除操作是将存储单元的内容由二进制的 0 变成 1，而写入操作恰好相反，是将存储单元的内容由二进制的 1 变成 0。擦除、写入操作都是通过设置或清除 Flash 控制寄存器(FLCR)中的某个或某些位来完成的。

Flash 在片内是以页(page)和行(row)为单位组织的。页和行的大小(字节数)随整个 Flash 的大小变化而变化，而页的大小始终为行的两倍。例如 MC68HC908GP32 内含 32kB 的 Flash，每页的大小为 128B，每行的大小为 64B；而 MC68HC908JL3 片内 Flash 仅有 4kB，每页和每行的大小也分别变为 64B 和 32B。最新推出的 MC68HC908LD64 将片内 Flash 分为了两个区域($0C00～$0FFF 和 $1000～$F9FF)，而每个区域对页的大小定义都是不同的(分别为 128B 和 512B)。对 Flash 的擦除操作都是以页为基础的，而写入操作则是以行或页为基础的(详见 Flash 的编程步骤)。

5.2.2 闪速存储器的编程寄存器

与 Flash 编程操作有关的寄存器有两个：Flash 控制寄存器(FLCR)和 Flash 块保护寄存器(FLBPR)。下面将详细讲述这些寄存器的主要功能和使用方法。

1. Flash 控制寄存器(FLCR)

如图 5-1 所示的 Flash 控制寄存器用于控制对 Flash 的擦除和写入编程操作。其中：

位	7	6	5	4	3	2	1	0	
$FE08	0	0	0	0	HVEN	MASS	ERASE	PGM	FLCR
复位	0	0	0	0	0	0	0	0	

图 5-1　Flash 控制寄存器

(1) HVEN 为高压允许位。当 PGM＝1 或 ERASE＝1 时，用户可以通过设置 HVEN 将来自片内电荷泵的高压加到 Flash 阵列上。

1＝打开电荷泵并将高电压加到 Flash 阵列上。

0＝撤除 Flash 阵列上的高电压并关闭电荷泵。

(2) MASS 为整体擦除控制位。该位在 ERASE＝1 时有效，用于选择 Flash 擦除操作方式：整体擦除或页擦除。

1＝选择整体擦除方式。

0＝选择页擦除方式。

(3) ERASE 为擦除操作控制位。该位用于设置 Flash 编程操作为擦除操作。ERASE 位与 PGM 位之间存在互锁关系，无法同时被设置为 1。

1＝选择擦除操作。

0＝不选择擦除操作。

(4) PGM 为写入操作控制位。该位用于设置 Flash 编程操作为写入操作。

1＝选择写入操作。

0＝不选择写入操作。

2. Flash 块保护寄存器(FLBPR)

Flash 块保护寄存器如图 5-2 所示，图中 U＝不受复位影响，出厂时设置为 1；BPR[7…0] 是 Flash 块保护寄存器位。Flash 块保护寄存器用于设定被保护的 Flash 区域，它本身也是一个 Flash 字节。当 Flash 处于保护状态时，擦除和写入操作在用户方式下都是受限制的，HVEN 将无法正常置位。实际上，Flash 块保护寄存器设定的只是保护区域的起始地址，因为保护区域的结束地址始终为 Flash 存储区的结束地址($FFFF)。

位	7	6	5	4	3	2	1	0	
$FF7E	BPR7	BPR6	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0	FLBPR
复位	U	U	U	U	U	U	U	U	

图 5-2　Flash 块保护寄存器

如果 FLBPR 的存储内容为 0，整个 Flash 存储区都受到保护；如果 FLBPR 的存储内容为 $FF，则整个 Flash 存储区都可以被擦除或写入。

需要注意的是，只有当微控制器处于用户方式下时，对 FLBPR 本身和 Flash 保护区域的擦写操作保护才是有效的。

FLBPR 的 8 位设置了 Flash 保护区域起始地址的高 8 位。需要注意的是，这 8 位并不是 16 位起始地址的最高 8 位，它们的具体位置取决于特定微控制器的 Flash 容量。例

如,对 MC68HC908GP32 而言,它们设定的是 16 位起始地址的第 14~7 位(第 15 位恒为 1)(见表 5-2);而对 MC68HC908JL3 而言,它们设定的是 16 位起始地址的第 12~6 位(第 15~13 位恒为 1,BPR0 位没有使用)(见表 5-3)。

表 5-2 MC68HC908GP32 Flash 块保护寄存器设置

FLBPR 内容(十六进制)	受保护的 Flash 区域
00	$8000 ~ $FFFF(整个 Flash 区域)
01	$8080 ~ $FFFF
02	$8100 ~ $FFFF
…	…
FE	$FF00 ~ $FFFF
FF	整个 Flash 区域都不受保护

表 5-3 MC68HC908JL3 Flash 块保护寄存器设置

FLBPR 内容(十六进制)	受保护的 Flash 区域
00 ~ 60	$EC00 ~ $FFFF(整个 Flash 区域)
62	$EC40 ~ $FFFF
64	$EC80 ~ $FFFF
68	$ECC0 ~ $FFFF
…	…
DE	$FBC0 ~ $FFFF
…	…
FE	$FFC0 ~ $FFFF
FF	整个 Flash 区域都不受保护

5.2.3 闪速存储器的编程步骤

对 Flash 进行擦除或写入操作需要遵循一定的时序和步骤。对于 MC68HC908 系列的微控制器,这些步骤都是一样的,但时序要求可能略有不同,用户应根据具体情况参考相应的芯片手册。

1. 页擦除操作

(1) 置 ERASE 位为 1,清 MASS 位为 0;
(2) 读 Flash 块保护寄存器;
(3) 向被擦除的 Flash 页内任意地址写入任意值;
(4) 延时 Tnvs;

(5) 置 HVEN 位为 1；
(6) 延时 Terase；
(7) 清 ERASE 位为 0；
(8) 延时 Tnvh；
(9) 清 HVEN 位为 0；
(10) 延时 Trcv 后，Flash 存储区可以被正常读取。

2. 整体擦除操作

(1) 置 ERASE 位和 MASS 位为 1；
(2) 读 Flash 块保护寄存器；
(3) 向 Flash 存储区内任意地址写入任意值；
(4) 延时 Tnvs；
(5) 置 HVEN 位为 1；
(6) 延时 Tmerase；
(7) 清 ERASE 位为 0；
(8) 延时 Tnvhl；
(9) 清 HVEN 位为 0；
(10) 延时 Trcv 后，Flash 存储区可以被正常读取。

3. 写入操作

在 MC68HC908 系列的众多微控制器中，有的以页为基础进行写入操作，如 MC68HC908GP32 等，有的以行为基础进行写入操作，如 MC68HC908JL3 和 MC68HC908JB8 等。用户在编程时需要根据实际选用的微控制器型号查阅相应的芯片手册。

(1) 置 PGM 位为 1；
(2) 读 Flash 块保护寄存器；
(3) 向即将被写入的 Flash 页（或行）内（以 MC68HC908GP32 为例，即 $XX00～$XX7F，或 $XX80～$XXFF）的任意 Flash 单元写入任意值。本步骤选定了即将被写入的 Flash 页，在接下来的步骤（7）中，编程数据写入的目标地址必须都存在于这一 Flash 页内。
(4) 延时 Tnvs；
(5) 置 HVEN 位为 1；
(6) 延时 Tpgs；
(7) 向页内目标地址写入编程数据；
(8) 延时 Tprog；
(9) 重复（7）、（8），直至同一页内各字节编程完毕；
(10) 清 PGM 位为 0；
(11) 延时 Tnvh；

(12) 清 HVEN 位为 0;

(13) 延时 Trcv 以后,Flash 存储区可以被正常读取。

表 5-4 是 MC68HC908GP32 的 Flash 编程的各项参数,其他型号的微控制器参数请参阅相应的芯片手册。其中,Terase 和 Tmerase 的最小时间也是擦除操作的最佳时间。如果擦除时间长于该最小值,Flash 的寿命会受到影响。Thv 指的是在下一次擦除操作执行以前,该 Flash 行在写入编程操作中被加上高电压的累积时间。

表 5-4　MC68HC908GP32 的 Flash 编程参数

参　　数	符号	最小值	最大值	单位
Flash 行大小	—	64	64	B
Flash 页大小	—	128	128	B
Flash 读写周期	Fread	32	8400	kHz
Flash 页擦除时间	Terase	1	—	ms
Flash 整体擦除时间	Tmerase	4	—	ms
Flash 从设置 PGM 或 ERASE 位到 HVEN 建立时间	Tnvs	10	—	μs
Flash 高电压维持时间(页擦除时)	Tnvh	5	—	μs
Flash 高电压维持时间(整体擦除时)	Tnvhl	100	—	μs
Flash 写入编程维持时间	Tpgs	5	—	ns
Flash 写入编程时间	Tprog	30	40	μs
Flash 返回只读状态时间	Trcv	1	—	μs
Flash 累积高电压时间	Thv	—	25	ms
Flash 行擦除次数	—	10000	—	次
Flash 行写入次数	—	10000	—	次
Flash 数据保持时间	—	10	—	年

5.2.4　闪速存储器的编程范例

下面将以 MC68HC908GP32 为例,讲述如何用汇编语言实现对 Flash 的擦除和写入编程。

1. 常量定义

```
FLCR     EQU    $FE08
FLBPR    EQU    $FF7E
C10US    EQU    $FFFE
C30US    EQU    $FFF8
C50US    EQU    $FFF1
```

```
C1MS        EQU     $FEC8
```

其中,FLCR 是 Flash 控制寄存器的常量定义,FLBPR 是块保护寄存器的常量定义,C10US、C30US、C50US 和 C1MS 用于延时例程,详见延时例程的介绍。

2. 延时例程

为了满足 Flash 编程操作中的时序需要,用户可以使用下面的例程进行必要的程序延时。

```
NULLCYCLE:
    AIX     #1              ; 2 个周期
    CPHX    #0              ; 3 个周期
    BNE     NULLCYCLE       ; 3 个周期
    RTS                     ; 4 个周期
```

该例程不断为 H:X 存储内容加 1,直至 H:X 存储内容为 0,所以 H:X 的初始存储内容直接决定了例程内部循环次数,即例程运行时间,以达到程序延时的效果。调用方法如下(以延时 1ms 为例):

```
LDHX        #C1MS           ; 3 个周期
BSR         NULLCYCLE       ; 4 个周期
```

如果循环次数为 n,则整个延时步骤耗费的总线周期为

3(LDHX)+ 4(BSR)+(2(AIX)+ 3(CPHX)+ 3(BNE)) * n + 4(RTS)

本范例工作的总线频率为 2.4576MHz,每个周期为 0.4μs,所以 10μs 需要 25 个周期,n=2(H:X= $FFFE);30μs 需要 75 个周期,$n$=8(H:X= $FFF8);50μs 需要 125 个周期,$n$=15(H:X= $FFF1);1ms 需要 2500 个周期,$n$=312(H:X= $FEC8)。

3. 整体擦除例程

下面是整体擦除例程(为了程序简便,在 10μs 以下的延时统一处理为 10μs 延时)。读者可以参照本书第 5.2.3 的内容阅读以下代码,注释中的步骤标号和本书第 5.2.3 中的讲述一一对应(页擦除例程和写入例程也是如此)。

```
SUB_MERASE:
    LDA     #6
    STA     FLCR            ; 第 1 步
    LDA     FLBPR           ; 第 2 步
    STA     $8000           ; 第 3 步
    LDHX    #C10US          ; 第 4 步
    BSR     NULLCYCLE
    LDA     #0E             ; 第 5 步
    STA     FLCR
    LDHX    #C1MS           ; 第 6 步
```

```
    BSR     NULLCYCLE
    LDHX    #C1MS
    BSR     NULLCYCLE
    LDHX    #C1MS
    BSR     NULLCYCLE
    LDHX    #C1MS
    BSR     NULLCYCLE
    LDA     #0C             ；第 7 步
    STA     FLCR
    LDHX    #C10US          ；第 8 步
    BSR     NULLCYCLE
    CLRA                    ；第 9 步
    STA     FLCR
    LDHX    #C10US          ；第 10 步
    BSR     NULLCYCLE
    RTS
```

4. 页擦除例程

页擦除例程如下(为了程序简便,在 10μs 以下的延时统一处理为 10μs 延时)：

```
*
BUFF        RMB     4
*
SUB_ERASE：
    LDA     #2              ；第 1 步
    STA     FLCR
    LDA     FLBPR           ；第 2 步
    JSR     BUFF            ；第 3 步
    LDHX    #C10US          ；第 4 步
    BSR     NULLCYCLE
    LDA     #0A;            ；第 5 步
    STA     FLCR
    LDHX    #C1MS           ；第 6 步
    BSR     NULLCYCLE
    LDA     #0C             ；第 7 步
    STA     FLCR
    LDHX    #C10US          ；第 8 步
    BSR     NULLCYCLE
    CLRA                    ；第 9 步
    STA     FLCR
    LDHX    #C10US          ；第 10 步
    BSR     NULLCYCLE
```

```
RTS
```

需要注意的是"JSR BUFF"语句，在这里使用它有一个前提，就是用户已经对以BUFF为起始地址的4个字节做了必要的初始化，这是为了使本例程能适用于不同Flash页的擦除操作。这4个字节实际上构成了一个很小的例程，完成在特定地址写入数据的功能。具体实现方法是由用户向第1个字节写入"STA <十六位地址>"语句的机器码$C7，向第2和第3个字节写入十六位地址(位于需要擦除的Flash页内)，最后向第4个字节写入"RTS"语句的机器码$81。

5. 写入例程

下面先介绍Flash写入(包括校验)例程使用的变量定义。

```
FlashFLAG    RMB    1
B_ERROR      EQU    0
BUFFCOU      RMB    2
COUNTE1      RMB    1
COUNTE2      RMB    1
PRADDR       RMB    2
TEMP         RMB    2
```

其中，FlashFLAG用于记录Flash写入操作的状态，它的第0位(B_ERROR)用于标志写入操作是否失败(0=成功，1=失败)。以BUFFCOU为起始地址的两个字节用于存储数据源缓冲区的起始地址，而COUNTE1则记录了需要从数据源缓冲区复制到Flash中的字节数。以PRADDR为起始地址的两个字节用于存储Flash目标写入区域的起始地址。COUNTE2和TEMP分别是COUNTE1和BUFFCOU的临时存储备份，其中COUNTE2还用于写入操作后的字节校验。

下面是具体的写入例程，其过程是将BUFFCOU指向的数据源缓冲区中的COUNTE1个字节一个个地复制到PRADDR指向的Flash目标写入区域中去(为了程序简便，在10μs以下的延时统一处理为10μs延时)。

```
SUB_PROGRAM:
    MOV     COUNTE1,COUNTE2     ；将 COUNTE1 和 BUFFCOU 备份
    LDHX    BUFFCOU
    STHX    TEMP
    LDA     #1;                 ；第 1 步
    STA     FLCR
    LDA     FLBPR               ；第 2 步
    LDHX    PRADDR              ；第 3 步
    STA     ,X
    LDHX    #C10US              ；第 4 步
    BSR     NULLCYCLE
    LDA     #9                  ；第 5 步
```

```
    STA     FLCR
    LDHX    #C10US          ;第 6 步
    BSR     NULLCYCLE
    LDHX    PRADDR          ;第 7 步
CYCLE_PR:
    PSHH                    ;保存目标写入地址
    PSHX
    LDHX    BUFFCOU         ;提取源数据
    LDA     ,X
    AIX     #1              ;将 BUFFCOU 指向下一个源数据
    STHX    BUFFCOU
    PULX                    ;取出目标写入地址到 H:X
    PULH
    STA     ,X              ;向目标地址写入数据
    AIX     #1              ;将 H:X 指向下一个目标地址
    PSHH                    ;保存目标写入地址
    PSHX
    LDHX    #C30US          ;第 8 步
    BSR     NULLCYCLE
    PULX                    ;取出目标写入地址到 H:X
    PULH
    DEC     COUNTE1         ;第 9 步
    BNE     CYCLE_PR
    LDA     #8              ;第 10 步
    STA     FLCR
    LDHX    #C10US          ;第 11 步
    BSR     NULLCYCLE
    CLRA                    ;第 12 步
    STA     FLCR
    LDHX    #C10US          ;第 13 步
    BSR     NULLCYCLE
```

下面是完成校验功能的代码。

```
    LDHX    TEMP            ;恢复 BUFFCOU 数值,指向缓冲区首字节
    STHX    BUFFCOU
    LDHX    PRADDR          ;取出目标写入地址到 H:X
VERIFY_PR:
    PSHX                    ;保存目标写入地址
    PSHH
    LDHX    BUFFCOU         ;提取源数据
    LDA     ,X
    AIX     #1              ;将 BUFFCOU 指向下一个源数据
```

```
    STHX    BUFFCOU
    PULH                        ; 取出目标写入地址到 H:X
    PULX
    CMP     ,X                  ; 比较源数据与目标地址中写入的数据
    BNE     ERROR_PR            ; 如果不同,则跳转到出错处理语句
    AIX     #1                  ; 如果相同,继续将 H:X 指向下一个目标地址
    DEC     COUNTE2             ; 递减 COUNTE2
    BNE     VERIFY_PR           ; 如果为 0 则校验完毕
    RTS
ERROR_PR:
    BSET    B_ERROR,FlashFLAG   ; 如果出错,置上标志位
    RTS
```

以上例程已经完全实现了对 Flash 进行擦除、写入编程的所有步骤。只要用户在恰当的环境中以恰当的方法调用它们,就能够很方便地实现对 Flash 的在线编程。下面的介绍就是向用户说明如何去正确地调用 Flash 擦写例程。

5.3 闪速存储器的编程模式

5.3.1 闪速存储器编程模式概述

MC68HC908 系列微控制器的片内 Flash 可以在两种模式下进行在线编程:监控模式和用户模式。在这两种模式下,对 Flash 进行编程操作的代码都是相同的(如本书 5.2.4节所述),唯一的区别仅仅在于调用这些例程的方式和环境。

Motorola 推出的 MC68HC908 系列微控制器都可以工作在监控方式下。只要微控制器的复位向量($FFFE~$FFFF)内容为"空"($FFFF),或是在微控制器复位时在特定引脚上加上特定的电压,就可以使微控制器在复位后进入监控工作方式。在监控方式下,微控制器内部的监控 ROM 程序开始工作,并通过半双工串行口为主机的远程控制提供服务。主机程序可以利用监控 ROM 提供的少数几个指令对微控制器内部地址进行读取、写入等基本操作,包括下载程序到 RAM 中并执行。在此基础上,主机可以通过主机程序或是下载到 RAM 中的程序完成对 Flash 编程所需的一系列操作(如本书 5.2 节所述)。

同时,在微控制器正常工作的过程中,程序也可在用户模式下对 FLASH 进行编程操作。此时只有对 Flash 进行编程写入或擦除的程序(见本书第 5.2.4 节)才是必须的,而不需要用户提供任何外部硬件条件(包括特定引脚电压和主机的存在)。如果需要,微控制器程序可以通过 SCI 串口从主机获得某些命令或数据,但开发者必须为此编写相应的串口通信和命令服务程序。

两种模式各有优缺点。监控模式需要外部硬件支持,但不需要微控制器内部程序的存在,所以适合对新出厂芯片进行编程写入,或是对芯片进行整体擦除或写入;用户模式

可以在微控制器正常工作时进入，所以常用在程序运行过程中对部分 Flash 单元进行修改，特别适合于目标系统的动态程序更新和运行数据存储。目前监控模式常被仿真器和编程器采用，而在实际的工程应用中，开发者往往只需要考虑和实现用户模式下的 Flash 在线编程。本章将只讲述用户模式下的 Flash 编程方法，有关监控模式和监控 ROM 的介绍可以在第 11 章找到。

5.3.2 用户模式下的闪速存储器编程方法

在用户模式下，开发者需要按本书第 5.2.3 节讲述的步骤编写相应的 Flash 编程操作子例程（见本书第 5.2.4 节），然后在适当的时候由主流程调用。

开发者需要特别注意的是，这段完成编程步骤的子例程代码应该驻留在什么地方。在 Motorola 公司某些关于片内 Flash 的宣传资料中，曾指出只要子例程代码与被编程的目标地址不在同一操作单元（页）中，对 Flash 的任何操作就都是可行的。但在 Motorola 公司的许多技术文档中，却一再强调应将子例程在 RAM 中运行。实际上对于目前的 MC68HC908 系列微控制器而言，一旦 Flash 控制寄存器（FLCR）的 HVEN 位为 1，整个 Flash 都会被加上高于普通工作电压的编程电压，这时对 Flash 内任何字节的读取都是不稳定的（包括对程序执行代码的读取），也就是说，微控制器此时执行的代码有可能是完全错误的。而一旦微控制器执行了错误的指令，系统的复位就是不可避免的了。笔者在 MC68HC908GP32 的使用过程中发现，如果直接调用 Flash 中的编程子例程，在对 Flash 进行页擦除时，操作是成功的，但会导致微控制器自动复位，而在对 Flash 进行写入时，系统复位并导致操作完全失败。所以 Flash 编程子例程必须存在于 RAM 或监控 ROM 中。如果用户使用的是自行开发的 Flash 编程子例程（见本书第 5.2.4 节），可以在需要对 Flash 进行编程前将子例程复制到 RAM 中去，然后跳转到 RAM 中执行。

对于 RAM 区较少的微控制器，如 MC68HC908JL3，开发者必须利用监控 ROM 中固化的程序。一般来说，在这类微控制器的监控 ROM 中，都已经固化了标准的 Flash 编程操作子例程。Motorola 公司的相关技术文档对这些子例程的接口和使用方法都有详细的介绍。

第6章 定 时 器

Motorola 微控制器定时器是主管定时的系统。和 MC68HC05 系列微控制器不同，MC68HC08 系列微控制器可以提供多个独立的定时器，MC68HC908GP32 就有两个定时器。

定时器的核心是一个 16 位计数器，该计数器的计数频率由外部晶振时钟（也可以是由锁相环模块得到的总线时钟）经过预定的分频因子分频得到，相互独立的定时器可以使用不同的分频因子。同一个定时器的所有动作都以经过这个分频的频率作为参考，所以相互之间有确定的关系。从微控制器的角度看，真正的时间间隔被这个自由运行的定时计数所代替。在任何时候可以通过读取计数器的读数而判断经过了多长时间。

使用预置计数功能可以得到精确的溢出时间，可以在任何时候暂停或清除计数器的计数。通过定时器的溢出功能可以进行长时间的定时。

定时器具有定时溢出、输入捕捉、输出比较、PWM 脉冲输出、DMA 直接存储器访问等功能。定时器的多种功能可以应用于多个场合。如利用输入捕捉测量脉宽、测量频率，利用输出比较输出特定的波形、直接输出脉宽调制波。另外，使用输入捕捉配合输出比较来输出同步波形或延时。下面就以 MC68HC908GP32 为例，详细阐述定时器的结构、功能及应用。

6.1 定时器的结构及基本功能

MC68HC908GP32 定时器的功能结构图如图 6-1 所示。从这张结构图可以看出定时器具有以下特征：每个定时器具有两路独立的输入捕捉和输出比较通道；可以实现带缓冲和不带缓冲的脉宽调制波；可以得到 7 种可编程选择的分频因子；计数器可以自由运行计数或预置计数；可以停止或清除计数器计数；定时器溢出可以引发中断，每路通道的输入捕捉和输出比较也可以引发中断。

每一个定时器都由 2 个引腿和 11 个寄存器组成。在 MC68HC908GP32 的不同封装形式中，这两个引腿的布局可能是不同的，需要查阅具体的芯片说明。而在 40 腿的双列直插式封装中只提供了一个定时器的引腿。

6.1.1 定时器的寄存器

MC68HC908GP32 两个定时器的寄存器的设置和功能是完全相同的，只是寄存器的地址不同。下面介绍定时器 1 的 11 个寄存器的名称和地址。

定时器 1 状态和控制寄存器	T1SC	$ 0020
定时器 1 计数器高位寄存器	T1CNTH	$ 0021

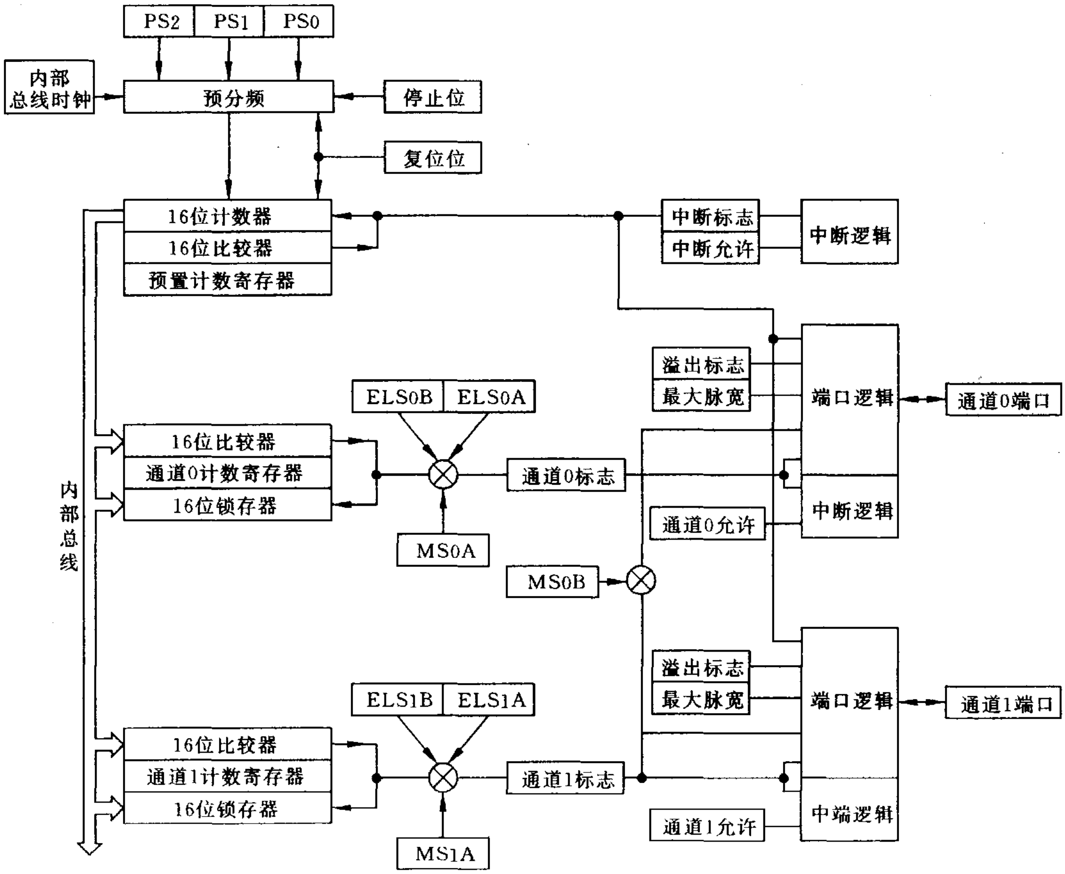


图 6-1　定时器功能框图

定时器 1 计数器低位寄存器	T1CNTL	$ 0022
定时器 1 预置计数高位寄存器	T1MODH	$ 0023
定时器 1 预置计数低位寄存器	T1MODL	$ 0024
定时器 1 通道 0 状态和控制寄存器	T1SC0	$ 0025
定时器 1 通道 0 高位寄存器	T1CH0H	$ 0026
定时器 1 通道 0 低位寄存器	T1CH0L	$ 0027
定时器 1 通道 1 状态和控制寄存器	T1SC1	$ 0028
定时器 1 通道 1 高位寄存器	T1CH1H	$ 0029
定时器 1 通道 1 低位寄存器	T1CH1L	$ 002A

在上面的各个寄存器的简写中，将 T1XX 替换为 T2XX 就成了定时器 2 的各个对应寄存器，其地址请查附录。在以后的程序中如果出现 T2XXX，就可以这样理解。

因为定时器为 16 位，所以相应的寄存器都使用了两个 8 位寄存器代表其高位和低位。

当读计数器的高字节时，低字节被自动锁存到缓冲区中。在读取低字节之前，这个缓冲区的内容将保持不变，即使多次读取高字节的内容，该缓冲区仍将保持第一次读取时的

低字节值。而读取低字节实际是对这个缓冲区进行操作,读低字节的操作将释放这个缓冲区使之和实际的计数器低字节保持同步变化。所以,一个完整的读计数器操作,只要读了高字节就应该读取低字节,即使此时低字节的计数对程序是没有用的。但只读低字节的操作却是允许的,执行该操作得到读取时刻的计数器低位字节值。

6.1.2 定时器溢出

16 位的自由运行计数器在微控制器复位结束后,从 $FFFC 开始连续计数,当达到最大值 $FFFF 时翻转到 $0000,同时置位状态寄存器的溢出位 TOF,然后重新开始计数。在微控制器正常工作时,这个计数器会自动计数,程序也可以停止或者清除计数器的计数。计数器的计数频率是由系统的内部总线频率和计数分频因子来决定的。分频电路的框图如图 6-2 所示。使用分频因子可以极大地扩大 16 位定时器的计数时间间隔,最大可以达到没有分频的时间间隔的 64 倍。这在实际使用中经常出现。如果总线频率为 2MHz,分频因子为 32,那么计数器每 16μs 计数一次。如果定时器是自由运行溢出,那么每隔 1.048s 将产生一次定时器溢出。如果允许定时器溢出中断,将会得到一次定时器的溢出中断服务。

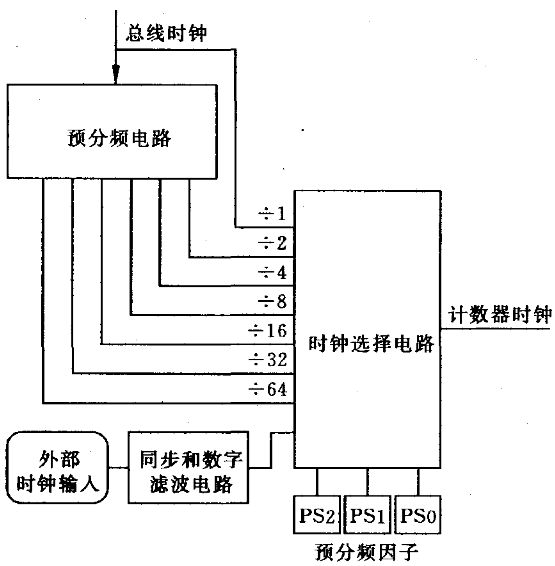


图 6-2 分频电路框图

6.1.3 定时器预置计数溢出

如果使用预置计数,则可以得到更精确和便于计算的溢出中断。预置计数寄存器是一个 16 位的寄存器,程序向这个寄存器写入一个确定的数值,则计数器每进行一次计数都会将计数和这个寄存器的值进行比较,如果相同就产生溢出,计数器翻转到零并重新开

始计数，也就是说，此时自由运行计数器没有计数到＄FFFF就产生溢出并重新计数。预置计数寄存器可以随时进行更改以得到不同时间间隔的定时器溢出间隔。仍然以上面的2MHz总线，分频因子为32为例，如果预置计数寄存器设置为＄F424，就可以得到1s间隔的定时器溢出，这对于时间计算就非常方便。

6.1.4 定时器的状态和控制寄存器

溢出标志和分频因子的设定等由定时器状态和控制寄存器来决定。这个寄存器的各位名称和作用如图6-3所示。

	位	7	6	5	4	3	2	1	0	
＄0020		TOF	TOIE	TSTP	TRST		PS2	PS1	PS0	T1SC
	复位	0	0	1	0	0	0	0	0	

图6-3　定时器状态和控制寄存器

1. TOF——定时器溢出标志位

该位用来标志定时器的溢出，当定时器的计数达到＄FFFF（自由计数运行状态）或预置计数器的数值时，定时器的计数清零，同时该位置1。读取定时器状态和控制寄存器，并向这一位写入零，可以清除该标志位。也就是使用如下的两条指令：

```
T1SC    EQU     $0020
LDA     T1SC                ;读取状态和控制寄存器
BCLR    7,T1SC              ;清除溢出标志位
```

该位虽然是可读写的，但由于该位是状态位，向该位写入1的操作没有任何的实际意义。另外需要注意，如果在清除该标志位之前又发生了定时器的溢出，那么这一次的清零无法完成。程序必须保证每一次溢出发生后，都要在下一次溢出发生之前清除该溢出标志位。在使用溢出中断的情况下就要求在中断返回之前必须清除溢出标志位，并且该中断的处理时间在最不利的情形下也不能超过一次计数溢出的时间。

2. TOIE——定时器溢出中断允许位

该位用来标志是否允许定时器溢出中断。该位置为1表示允许溢出中断，当发生溢出时，系统会执行溢出中断处理程序。该位置为0表示不允许溢出中断。该位是可以读写的，读取的数值就是该位的实际值。

3. TSTP——定时器计数停止/恢复位

该位用来控制计数器的计数停止和恢复。该位置为1，定时器将保持当前计数，不再随计数脉冲递增。该位置为0，定时器将恢复计数，从停止的计数继续递增。

```
T1SC        EQU     $0020
```

```
TSTP        EQU    5
;执行其他命令
BSET        TSTP,T1SC
;执行其他命令                    ;该段程序执行期间计数器保持计数不变
BCLR        TSTP,T1SC
```

需要注意,该位在系统复位后的默认值为 1,这时计数器是停止的。所以在程序中如果需要使用定时器必须首先清除该位。该位也是可读写的。

4. TRST——定时器计数清零

向该位写入 1 将清除定时器的计数和设定的分频因子,在定时器计数清零的时候将自动清除该位。也就是说该位通常总是为 0,写入 1 的操作只是引发了一次正跳变的脉冲。事实上对该位的读取值总是 0。

5. PS2～PS0——定时器分频因子设定位

这 3 位共同设定定时器的分频因子。3 位的不同组合有 8 种可能,使用了其中的 7 种来提供 7 个不同的分频因子。

这 3 位的值和设定的分频因子的对应关系如表 6-1。

表 6-1　定时器分频因子

PS2	PS1	PS0	分频因子
0	0	0	内部总线频率除以 1
0	0	1	内部总线频率除以 2
0	1	0	内部总线频率除以 4
0	1	1	内部总线频率除以 8
1	0	0	内部总线频率除以 16
1	0	1	内部总线频率除以 32
1	1	0	内部总线频率除以 64
1	1	1	没有使用

系统复位后,这 3 位的值是 000,系统使用没有分频的内部总线频率作为计数器的计数频率。

定时器的状态和控制寄存器的各位都有不同的作用,在设定时可以根据需要只对需要更改的某一位进行读写,也可以对整个寄存器进行读写。在系统初始化的时候一般按下述步骤进行状态和控制寄存器的设置。

(1) 根据需要确定各位的值;

(2) 由各个位的值得到整个寄存器的值;

(3) 将这个值写入控制和状态寄存器就可以完成各位的设定。

例　现在设定要求为：允许溢出中断；恢复定时器计数；计数频率为内部总线频率的8分频。

综合上述的要求，需要配置该寄存器的值为＄63，那么只需用以下两条语句就可以完成。

```
LDA     #$63
STA     T1SC
```

通常，在设置这些寄存器值的时候，习惯用二进制方法来表示写入的数值，这样更容易清楚地看出各位的设定。

6.2　输入捕捉

6.2.1　输入捕捉的概念

输入捕捉是 Motorola 微控制器定时器的基本功能，该功能用来监测外部的事件和输入信号。当外部事件发生或信号发生变化时，在指定的输入捕捉引脚上发生一个指定的沿跳变（可以指定该跳变是上升沿还是下降沿）。定时器捕捉到特定的沿跳变后，把自由运行计数器当前的值锁存到输入捕捉寄存器。如果在输入捕捉控制寄存器中设定允许输入捕捉中断，系统会产生一次输入捕捉中断，利用中断处理软件可以得到事件发生的时刻或信号发生变化的时刻。输入捕捉模块的框图如图 6-4 所示。

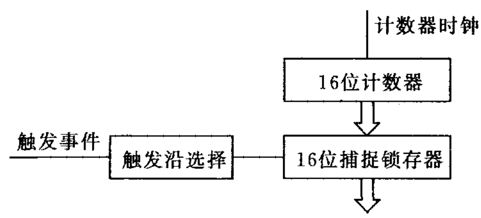


图 6-4　输入捕捉模块框图

通过记录输入信号的连续的沿跳变，就可以用软件算出输入信号的周期和脉宽。例如，为了测量周期，只要捕捉到两个相邻的上升或下降沿的时间，两者相减就可以得到周期；为了测量脉宽，就要记录相邻的两个不同极性的沿变化的时间。当测量的脉宽值小于定时器的溢出周期时，只需将两次的值直接相减（看成无符号数）。如果测量值大于定时器的溢出周期，那么在两次输入捕捉中断之间就会发生定时器计数的溢出翻转，这时直接将两个数相减就没有意义，需要考虑到定时器的溢出次数。

输入捕捉的另一个用途是配合输出比较来作延时。例如，当需要在一个外部事件发生一定的时间后，微控制器产生一个输出信号来控制某个操作。这就可以利用输入捕捉来记录外部事件发生的时刻，把这个事件加上一定的延时值送到输出比较寄存器，并允许输出比较功能，这样就可以达到上述的目的。

6.2.2 输入捕捉操作

输入捕捉功能由16位的锁存器、输入边沿探测逻辑和中断逻辑组成。当指定的跳变沿出现在输入捕捉引脚上时，锁存器锁存计数器的计数值。跳变沿的极性选择、是否允许输入捕捉中断都在输入捕捉控制寄存器中设定。

实际上MC68HC08系列的定时器中，每一个定时器都有两路独立的输入捕捉/输出比较通道，每一路通道都既可以作为输入捕捉，又可以作为输出比较。如果作为输入捕捉，该通道的控制寄存器就是输入捕捉控制寄存器；如果作为输出比较，该寄存器就作为输出比较控制寄存器。也就是说，输入捕捉寄存器和输出比较寄存器只是同一个寄存器在作为不同用途时的不同称呼。

输入捕捉寄存器是只读寄存器，在每次发生了指定的沿跳变时，都会自动地将自由运行计数器的内容传给输入捕捉寄存器。输入捕捉寄存器是一个16位的寄存器，分为高字节和低字节，在读取的时候要分别读取。为了防止两次读取之间该寄存器的内容发生变化而产生虚假的输入捕捉计数值，系统会在读取高字节时锁存低字节的内容，这时即使又发生特定的沿跳变，输入捕捉寄存器的内容也不会改变。此后对低字节的读取将释放输入捕捉寄存器，但只读取输入捕捉寄存器的低字节将不影响输入捕捉寄存器。所以，如果要读取整个输入捕捉寄存器，必须先读高字节，再读低字节。

由于在读取低字节之前输入捕捉寄存器是"失效"的，而读取低字节的指令以及中断的进入和返回也会花费一定的时间，所以用输入捕捉来测量输入信号时，对输入信号的频率有一个上限，大于这个频率的信号就不能正确测量。

6.2.3 输入捕捉的控制寄存器

输入捕捉控制寄存器是某个定时器在作为输入捕捉功能时的称呼，在用作输出比较时就是输出比较控制寄存器。在这两种情形下，这个控制寄存器的很多位的作用都是一样的，在后面的输出比较中就不解释了。

通道0的输入捕捉寄存器各位名称和功能如图6-5所示。

	位	7	6	5	4	3	2	1	0	
$0025		CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	TOV0	CH0MAX	T0SC
	复位	1	1	1	1	1	1	1	1	

图6-5 通道0输入捕捉/输出比较控制寄存器

通道1的输入捕捉寄存器各位名称和功能如图6-6所示。

	位	7	6	5	4	3	2	1	0	
$0028		CH1F	CH1IE		MS1A	ELS1B	ELS1A	TOV1	CH1MAX	T1SC
	复位	1	1	1	1	1	1	1	1	

图6-6 通道1输入捕捉/输出比较控制寄存器

1. CHxF——通道标志位

该位用来标志定时器通道发生了输入捕捉。在发生输入捕捉时，输入捕捉寄存器记录此时定时器的计数，并将该位置 1。读取该寄存器并向这一位写入零可以清除该标志位。

2. ChxIE——通道中断允许位

该位用来设置是否允许发生输入捕捉中断。该位置 1，在发生输入捕捉时会产生中断。在中断服务程序中可以进行相关的处理，通常在这个中断程序中应该包含清除通道标志位的指令。

3. MSxB，MsxA——模式选择位

每一个定时器都可以工作在输入捕捉，输出比较和 PWM 输出模式，这两位用来选择这些工作模式。因为在带缓冲的 PWM 模式下，需要两个通道同时使用，所以通道 1 没有 MSxB 选择位。模式选择位和下面的跳变沿/输出电平选择位共同决定通道的工作方式。

4. ELSxB，ELSxA——跳变沿/输出电平选择位

该位在输入捕捉时设定上升沿或下降沿捕捉，输出比较时设定输出高电平或低电平。每一位的设定和工作方式的关系见表 6-2。

表 6-2　通道控制寄存器控制位

MSxB 和 MSxA	ELSxB 和 ELSxA	工作模式	跳变沿/输出电平
X 0	0 0	预设输出电平	初始输出高电平
X 1	0 0		初始输出低电平
0 0	0 1	输入捕捉	上升沿捕捉
0 0	1 0		下降沿捕捉
0 0	1 1		跳变沿捕捉
0 1	0 1	输出比较	输出电平翻转
0 1	1 0		输出高电平
0 1	1 1		输出低电平
1 X	0 1	带缓冲的输出比较或带缓冲的 PWM 输出	输出电平翻转
1 X	1 0		输出高电平
1 X	1 1		输出低电平

5. TOVx——溢出翻转控制标志位

6. CHxMAX——通道最大占空比设置位

定时器通道用作输入捕捉时，这两位没有作用。在后面的输出比较中将讲述它们的作用。

6.2.4 输入捕捉编程实例

输入捕捉可以用来判断输入信号的周期和占空比，在很多场合是非常有用的。

例 6.1 在使用电话的时候，可以听到电话线路上的信号音。信号音的频率是有统一规定的。例如线路空闲音中心频率为 450Hz，误差为正负 25Hz，连续发送；线路忙音中心频率也为 450 Hz，误差为正负 25Hz，但是 0.35s 发送，0.35s 停止；而拨号回铃音中心频率仍为 450Hz，0.7s 发送，0.7s 停止。根据这些标准，通过将信号音滤波整形得到标准的方波就可以用输入捕捉来处理判断。

需要说明一下，默认的晶振频率为 8MHz，系统总线频率为 2MHz，时钟预分频因子为 32，所以参考时钟周期为 16μs。预置计数器的写入值为 625，所以时钟溢出中断为 10ms 一次，这是维护一个实时时钟的关键。

程序如下：

```
TELE_SIG_INTERRUPT:
    PSHH
    PSHX
    SEI                                 ;开中断
    LDA     T1SC0                       ;清除中断标志
    BCLR    7,T1SC0
    LDA     T1CH0H                      ;读取输入捕捉寄存器的值
    LDA     T1CH0L
    PSHA
    SUB     TELE_TEMP1                  ;与上一次的值相减并且取绝对值，得到两次输入捕捉
                                         之间时间间隔
    BHI     TEMP_LABEL
    COMA
TEMP_LABEL:
    BRCLR   TOVER,SYS_FLAG,N_FLOW
    BCLR    TIM_OVER,SYS_FLAG           ;如在两次输入捕捉之间发生时钟溢出，输入捕捉寄存
    ADD     #113T                       ;器的值也随着时钟清零。此时输入捕捉计数器低位
                                         丢失计数 113(由预置计数器低字节决定)，需要补回
                                         来。Tover 位在时钟溢出中断里置位
N_FLOW:                                 ;T(450Hz)=2222μs,2222μs/16μs=139
    CMP     #131T                       ;T(475Hz)=2105μs,2222μs/16μs=131
```

```
        BLO     TELE_SIG_INT_END
        CMP     #147T              ;T(425Hz)=2353μs,2353μS/16μs=147
        BHI     TELE_SIG_INT_END
        LDA     TELE_COUNT1
        ADD     #1
        STA     TELE_COUNT1
        LDA     TELE_COUNT2
        ADC     #0
        STA     TELE_COUNT2        ;一个16位计数器加一
TELE_SIG_INT_END:
        PULA
        STA     TELE_TEMPL         ;保存输入捕捉值,供下次比较用
        PULX
        PULH
        CLI
        RTI
```

6.3 输出比较

6.3.1 输出比较的概念

输出比较模块也是MC68HC908GP32定时器的基本组成部分。该模块用来在程序规定的时刻输出需要的电平,实现对外部电路的控制。自由运行计数器的值每递增一次,就和输出比较寄存器的值比较一次。当两个值相等的时候,输出比较模块置位定时器通道的状态和控制寄存器的CHxF位,并且在该通道的引脚上输出预先规定的电平。如果输出比较中断允许,还会产生一个定时中断。

使用输出比较的优势在于可以得到非常精确的输出时间间隔。硬件的比较功能不受其他中断的影响,而且对用户程序没有额外的负担。

输出比较最简单和最常用的场合就是产生一定间隔的脉冲。典型的应用实例就是实现软件的串行通信。用输入捕捉作为数据输入,而用输出比较作为数据输出。首先根据通信的波特率向输出比较寄存器写入延时的值,根据待传的数据位确定有效输出电平的高低。在输出比较中断处理例程中,重新更改输出比较寄存器的值,并根据下一位数据改写有效输出电平控制位。

6.3.2 输出比较操作

输出比较模块的逻辑结构如图6-8所示。

输出比较寄存器是一个16位寄存器,自由运行计数器的值每递增一次就和它比较一次,如果相等就置位CHxF位,并根据预先的设定输出高电平、低电平或翻转输出比较引

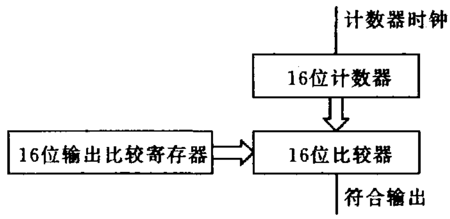


图 6-8　输出比较模块框图

腿的电平。

输出比较寄存器是16位的，通常会分两次写入。这时需要注意，在写入输出比较寄存器的高位字节后，为了防止此时的计数恰好等于自由运行计数器的值而产生不希望的输出比较动作，输出比较模块此时是被禁止的。只有在继续写入输出比较寄存器的低位字节后，输出比较模块才开始和自由运行计数器进行比较。所以，完整的设置输出比较功能应该包括对两个寄存器字节的写入。

输出比较寄存器和标志位在复位时被清零，在初始化输出比较功能时应该小心谨慎，一般采用以下的步骤。

(1) 写入输出比较寄存器的高位字节，禁止输出比较功能。

(2) 读取状态寄存器，然后立即向 CHxF 位写入 0 以清除该标志，因为如果不立即清除该标志，下一次比较成功又到来了，即便再写 0，这个标志也清不掉。

(3) 写入输出比较寄存器低位字节，使输出比较功能工作。

6.3.3　输出比较的控制寄存器

在前面已经提到，输出比较控制寄存器和输入捕捉控制寄存器是同一个寄存器在不同工作方式时的不同称呼。在用作输出比较时，该控制寄存器的很多位的功能和用作输入捕捉时是相同的。下面将有所不同的控制位稍做解释。

1. CHxF——通道标志位

此时该位用来标志定时器通道发生了输出比较，读取该寄存器并立即向这一位写入零可以清除该标志位。

2. ChxIE——通道中断允许位

该位此时用来设置是否允许发生输出比较中断。该位置 1，在发生输出比较时会产生中断。在中断服务程序中可以进行相关的处理，通常这个中断程序中应该包含清除通道标志位的语句。

模式选择位和输出电平选择位要根据具体的情况进行设置，而且通常在输出比较中断服务例程中还会更改输出电平选择位以得到所需电平。

3. TOVx——溢出翻转控制标志位

在定时器通道用作输出比较功能时，如果置位该位，在通道的定时器发生溢出时，输出比较模块会自动翻转输出电平。

在一个完整的脉冲中，必然有两次电平的跳变。这就需要两次进入输出比较处理例程，两次计算下一次输出比较的计数值。如果能够使脉冲的周期恰好等于定时器通道的溢出周期，并且置位溢出翻转控制标志位，那么在每次脉冲周期到的时候通道溢出功能将自动翻转输出电平而节省了一次输出比较的处理。这样做还有另一个优点，通常的输出比较处理例程中需要将当前计数器的值加上预期的下一次电平变化的时间间隔作为新的输出控制比较寄存器的值，而这个计算的过程是很费时的。如果使用溢出翻转，脉冲的周期就是定时器的溢出周期，则脉冲的占空时间也是一个确定的值，在输出比较中断中勿需计算可以直接填入这个预先确定的数值。

需要注意的是，置位溢出翻转位后，如果定时器溢出和输出比较同时发生，模块将优先处理定时器溢出事件。如果定时器溢出的处理例程比较耗时，输出比较将不能得到及时的处理。这在编写程序时一定要非常留意。

4. CHxMAX——通道最大占空比设置位

该位在定时器通道作为 PWM 输出的时候起控制作用，将在第 6.4 节中介绍。

6.3.4 不带缓冲的输出比较

MC68HC08 提供了两种输出比较的功能，它们是带缓冲和不带缓冲的输出比较。其中，不带缓冲的输出比较的功能和 MC68HC05 的输出比较功能相同，在更改下一次输出比较时需要重新写入输出比较计数器。使用不带缓冲的输出比较，两次输出比较之间的时间间隔就会因为重新写入输出比较计数器占用的时间而有一个下限，不可能小到几个总线周期的水平，这在应用中有一定的限制。

在对时间间隔要求不是非常严格的情况下，可以使用不带缓冲的输出比较。下面通过一个编程实例来说明不带缓冲的输出比较的使用。

例 6.2 利用不带缓冲的输出比较功能产生一个脉宽为 5s 的方波。使用 MC68HC908GP32 微控制器，外部晶振频率为 8MHz，内部总线频率为 2MHz，定时器预分频因子为 4，每一次计数时间为 2μs，每 65536 次计数，定时器溢出翻转一次，时间为 131.072ms。5s/131.072ms = 38 次定时器溢出，并且剩余 $25A0 次计数。

编程时使用定时器的通道 0，该通道将以 PORTD 的第四个引脚作为输出比较引脚，在该脚上接入一个发光二极管，可以看到二极管以 5s 的周期闪烁。如果在程序中同时对其他的 I/O 口的引脚进行翻转，也可以得到同样的结果。例子中同时对 PORTA 的第一个引脚进行了翻转。

程序如下：

```
PORTA             EQU     $00
```

```
DDRA        EQU    $04
T1CH0H      EQU    $26
T1CH0L      EQU    $27
T1SC        EQU    $20
T1SC0       EQU    $25
TCOUNT      EQU    $A0
TEMP        EQU    $A1
    ORG  $8000
INIT:
        LDA     #%00000001
        STA     T1SC
        MOV     #$FF,PORTA
        LDA     #%00000001
        STA     DDRA
        LDA     #%00010100
        STA     T1SC0
BEGIN
        LDA     PORTA
        EOR     #%00000001
        STA     PORTA
        LDA     T1CH0L
        ADD     #$A0
        STA     TEMP
        LDA     T1CH0H
        ADC     #$25
        STA     T1CH0H
        LDA     TEMP
        STA     T1CH0L
        MOV     #38,TCOUNT
LOOP1:  BRCLR   7,T1SC0,LOOP1
        LDA     T1CH0L
        LDA     T1SC0
        BCLR    7,T1SC0
        DBNZ    TCOUNT,LOOP1
        BRA     BEGIN
```

6.3.5 带缓冲的输出比较

前面讲到，使用不带缓冲的输出比较在实际使用中会带来一定的限制，输出脉冲的频率不能太高，而且脉冲的占空比也不能太小。而使用 MC68HC908GP32 的带缓冲的输出

比较功能就可以克服这个缺陷。

带缓冲的输出比较功能使用了同一定时器的两路通道，将这两路通道联合起来共同组成一个带缓冲的输出比较。该输出比较波形将出现在 TCH0 引脚上，而原来的 TCH1 引脚仍然可以作为普通的 I/O 引脚使用。

要使用带缓冲的输出比较功能，需要在初始化的时候置位定时器通道 0 的控制状态寄存器的 MS0B 位，并且向通道 0 的输出比较寄存器写入值。当发生了一次定时器溢出后，通道 1 也取得了对 TCH0 的输出控制权，此时两路通道将共同控制 TCH0 的输出，从而形成了一个带缓冲的输出比较器。

使用带缓冲的输出比较，控制输出比较的翻转的寄存器就不仅仅是某一个通道的输出比较寄存器，而是由两套输出比较寄存器来共同控制。这一点是带缓冲的输出比较的关键，所以要特别强调。这两套输出比较寄存器将交替控制输出比较的输出，具体来说向哪一套输出比较寄存器写入新的值，哪一套输出比较寄存器就取得了对比较输出的控制。而在下一次更改比较输出间隔的时候，必须写入另一套输出比较寄存器，使另一套输出比较寄存器在接下来的一个周期取得控制权，这样在更改输出比较周期时，可以有一套输出比较寄存器作为缓冲。被更改的输出比较寄存器和当前正在工作的输出比较寄存器是相互独立的，区别于不带缓冲的输出比较，所以能够克服不带缓冲的输出比较的不足。可以对照带缓冲的输出比较的框图 6-9 来理解这一点。

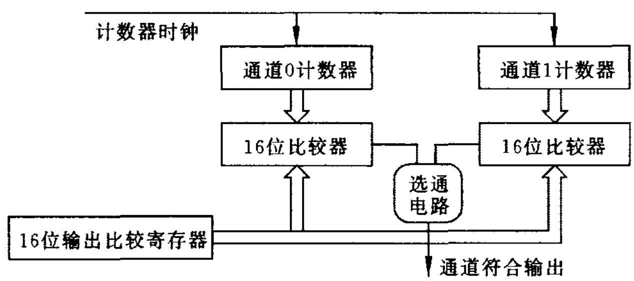


图 6-9 带缓冲的输出比较模块框图

不带缓冲的脉宽调制和带缓冲的脉宽调制也存在这种对比，情况比较类似，在下面的脉宽调制一节中将举例说明，读者可以参照此例来理解带缓冲的输出比较和不带缓冲的输出比较的对比。

6.4 脉宽调制输出

6.4.1 脉宽调制的概念

脉宽调制波是一种可以使用程序来控制波形占空比、周期、相位的波形。它在三相电机的驱动、D/A 变换等场合有普遍的应用。MC68HC908GP32 的定时模块也提供了脉宽调制的机制。

在前面曾经提到，定时器的每一个通道都可以设置为在溢出时自动翻转电平。合理地使用这个特性，定时器通道就可以产生脉宽调制波。预置计数寄存器的值将决定脉宽调制波的周期。当自由运行计数器的计数达到预置计数器的值时，该通道的输出将自动翻转。所以两次溢出之间的时间就是脉宽调制波的周期。

每一个通道的输出比较寄存器的值将决定脉宽调制波的占空比。因为在发生输出比较时，输出电平将根据每一个通道的控制状态寄存器的 ELSxA、ELSxB 位的设置而发生改变。这样在一次时钟溢出的周期内输出电平将发生两次翻转，而从发生输出比较到发生时钟溢出之间的时间就是脉宽调制波的占空比。脉宽调制 PWM 模块的框图如图 6-10 所示。

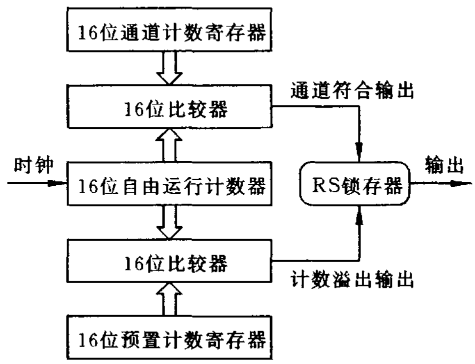


图 6-10　脉宽调制模块框图

当使用脉宽调制功能时，需要按照一定的顺序来初始化脉宽调制模块。

(1) 在定时器的状态和控制寄存器中，首先置位计数器停止位 TSTOP，然后置位计数器的复位位 TRST，重新开始计数器的计数。

(2) 在预置计数寄存器中写入希望的脉宽调制波的周期值。

(3) 在对应通道的输出比较寄存器中写入希望的脉宽调制波的脉宽值。

(4) 在对应通道的状态和控制寄存器中，首先根据是否是带缓冲的脉宽调制设置正确的 MSxB、MSxA 两位(不带缓冲的这两位置为 0 和 1，带缓冲的这两位置为 1 和 0)，然后置位溢出翻转位 TOVx，最后根据脉宽调制波的极性设置 ELSxB、ELSxA 两位。

同输出比较一样，MC68HC908GP32 的脉宽调制波也分为带缓冲的脉宽调制波和不带缓冲的脉宽调制波两种。下面分别解释这两种情况。

6.4.2　不带缓冲的脉宽调制波

任何一个定时器通道都能够产生不带缓冲的脉宽调制波，具体来说是在定时器预置计数寄存器中写入需要的脉宽调制波的周期，在希望输出波形的通道的输出比较寄存器中写入需要的脉宽调制波的脉宽。

之所以说这种方式是不带缓冲的，是因为如果需要更改脉宽调制波的脉宽，就必须重新改写当前定时器通道的输出比较寄存器。同前面讲到的不带缓冲的输出比较一样，如

果重新写入输出比较寄存器的时机不恰当，将导致脉宽调制输出一到两个周期的不正常。例如，将原有的脉宽调窄，需要写入更小的脉宽值。如果写入这个新的值时自由运行计数器已经超过了该新值，但还没有到达原有的旧值，那么在这个周期中将不会产生任何的输出比较。同样使用时钟溢出中断服务例程来写入新的更小的脉宽值也有可能丢失输出比较。

为了理解这种情形，下面通过图示来说明。

首先假定定时器通道被设置为发生输出比较时输出的引脚信号清零，同时允许计数器溢出时输出翻转。为了更改脉宽调制波形的脉宽，就要在输出比较中断或定时器溢出中断中更改脉宽值。对这两种情况将分别探讨。

如果使用时钟溢出中断来更改脉宽，将可能出现图 6-11 所示的情况。当定时器溢出时，中断服务请求也同时发生，但中断服务例程需要一段时间来为下一次输出比较操作设置脉宽值。这段时间延迟还同时取决于当时有没有其他中断正在被处理。所以，在一个非常繁忙的应用中，这段时间延迟实际上是无法预料的。而用户一般都需要去考虑最坏的情况。

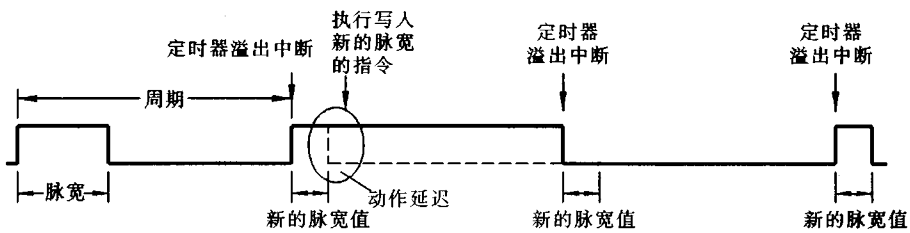


图 6-11　使用时钟溢出中断更改 PWM 脉宽

在这个例子中，假设溢出中断服务例程为下一次时钟通道变化（比如一次输出比较）花费了太多的时间去计算和设置。在执行写入新的脉宽值的指令的时候，自由运行计数器的值已经超过了设置的新的脉宽值，实际本周期中输出比较操作没有被执行，而输出引脚的状态也没有改变。这个状态一直维持到下一个计数器溢出中断，输出电平被翻转。现在输出比较操作的设置已经是新的脉宽值。然而，由于输出比较操作是设定为将引脚信号清零，所以引脚信号实际上并不改变。最后，只有在下一个计数器溢出中断完成后才得到想要的脉宽调制波。

可以看出，当使用计数器溢出中断服务例程来同步改变 PWM 的脉宽时，如果脉宽值改得很小（由应用决定），有可能产生意想不到的操作，信号出现一个 100％占空比和一个 0％占空比的周期后才出现设定的波形。每次将脉宽值改到一个很小的值时都会出现这种情况。所以，使用计数器溢出中断并不能避免误动作的情况。

此外，还可以用输出比较中断来同步改变 PWM 的脉宽值。正如图 6-8 所示，这个中断会在计数器溢出前发生。如果把脉宽值改得很大，和前面一样，当发生输出比较时，中断服务请求也同时发生，而中断服务例程需要一段时间延迟来为下一次输出比较操作设置时钟通道。但这个延迟肯定能在下一次 PWM 周期之前完成。实际上，这次计算和设置操作很可能会导致在同一个 PWM 周期里出现第二次输出比较中断。但它并不会对输

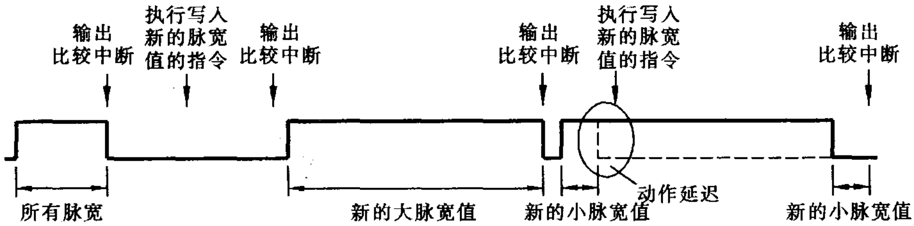


图 6-12 使用输出比较中断改变 PWM 脉宽

出引脚产生不利的影响，而且这第二次中断服务请求能够安全地被忽略掉。在接下来的计数器溢出中断中，溢出翻转功能使输出翻转。接下去的输出比较也会执行正确的操作。一直到现在为止，一切都是正常的。

如果再把 PWM 的脉宽编程为一个很小的值，又会发生什么情况呢？我们发现遇见了和前面一样的问题。由于潜在的中断响应时间的影响，新脉宽值的计算花费了过长的时间。仍然和前面的分析一样，信号会出现一个 100%占空比和一个 0%占空比的周期，然后才正常出现设定的波形。每次将脉宽值由一个很大的值改到一个很小的值时都会出现这种情况。仔细地分析一下，会发现如果只是在从一个长脉冲变化到一个短脉冲时在它们之间插入了一个 100%和一个 0%占空比的周期而导致错误的话，这实际上是可以接受的。然而，关键在于最坏情况下的中断响应时间，如果系统处理很多的中断源，那么一个很长的响应时间会使得这类操作让人无法接受。

将两个方法进行比较可知，使用输出比较中断只在从一个很大的脉宽值变化到一个很小的脉宽值的时候才会出现上述问题，而使用计数器溢出中断则不论脉宽改变前的值是多少，只要是将脉宽改变为一个很小的值都会出现上述情况，所以使用输出比较中断会优于使用计数器溢出中断，但它们都不能避免出现误操作的可能。

按照一定的编程规则可以尽量降低这种情况出现的概率。如果是更改更小的脉宽值，首先允许输出比较中断，并且在输出比较中断中写入这个新的脉宽值。如果是更改更大的脉宽值，首先允许定时器通道溢出中断，并且在定时器通道溢出中断中写入这个新的脉宽值。

要根本解决上述问题，必须使用带缓冲的输出比较功能。

6.4.3 带缓冲的脉宽调制波

使用带缓冲的脉宽调制波可以避免不带缓冲的脉宽调制的诸多限制。在这种方式下，一个定时器的两个通道联合起来交替控制原来通道 0 的脉宽调制输出，从而构成一个带缓冲的脉宽调制输出。该输出比较波形将出现在 TCH0 引脚上，而原来的 TCH1 引脚仍然可以作为一般的 I/O 引脚使用。该模块的功能框图如图 6-13 所示。

要使用带缓冲的脉宽调制功能，需要在初始化的时候置位定时器通道 0 的控制状态寄存器的 MS0B 位，并且向通道 0 的输出比较寄存器写入值。当发生了一次定时器溢出后，通道 1 就取得了对 TCH0 的输出控制权，此时两路通道将共同控制 TCH0 的输出，从

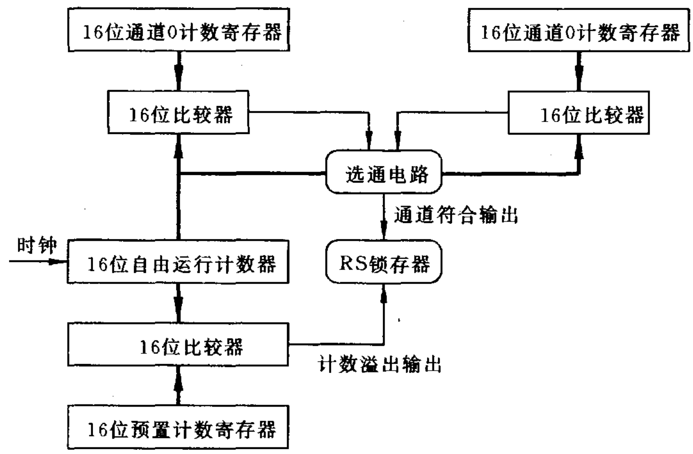


图 6-13　带缓冲的脉宽调制模块框图

而形成一个带缓冲的脉宽调制模块。这和带缓冲的输出比较功能类似。

使用带缓冲的脉宽调制，决定脉宽值的寄存器是最近被改变的寄存器。也就是说，如果向通道 0 的寄存器写入了新的脉宽值，那么在接下来的一个周期中，这个脉宽值将被使用。如果现在要更改脉宽值，就必须向通道 1 的寄存器写入新的脉宽值。在接下来的一个周期中，这个新的值将发生作用。由于在写入寄存器的时候，这个寄存器并不是当前使用的寄存器，对当前周期的波形没有任何影响，而在下一个周期中才同步地发生作用，所以这就形成了一个缓冲，避免了不带缓冲的脉宽调制的弊端。

需要注意两点：

(1) 使用了脉宽调制功能，就不能使用输出比较翻转。因为脉宽调制默认时钟溢出时输出翻转，如果输出比较发生时输出也翻转，那么整个脉宽调制波就有可能出现两种截然相反的极性。

(2) 使用带缓冲的脉宽调制波，就不能向当前活动的寄存器执行写入命令。因为如果这样，就和不带缓冲的脉宽调制波一样了，也就失去了带缓冲的脉宽调制的优点。

第 7 章　时钟发生模块

本章将详细介绍 MC68HC08 单片机中的时钟发生模块(clock generator module, CGM)的结构、功能和编程实现,并结合具体应用举出实例。时钟发生模块中的重要电路是锁相环(phase locked loop,PLL)频率合成器。本章先介绍锁相环频率合成器的基本概念,再按从一般到具体的顺序介绍 MC68HC08 和 MC68HC08GP32 中的时钟发生模块。

7.1　通用锁相环频率合成器简介

锁相环频率合成技术在通信、雷达、导航、宇航、遥控遥测、电子技术测量等领域都有广泛的应用。锁相环技术是实现相位自动控制的一门科学,起源于人们对各种电子设备中的频率源的稳定度和频率精度的要求,它具有宽频带范围和多波道的特点。

所谓频率合成技术,就是利用一个或几个具有高稳定度和高精度的频率源(一般由晶体振荡器产生),通过对它们进行加减(混频)、乘(倍频)、除(分频)运算,产生大量的具有相同频率稳定度和频率精度的频率信号。常用的频率合成技术有两种:直接频率合成和间接频率合成。

直接频率合成技术是将一个或几个晶体振荡器产生的频率信号通过谐波发生器产生一系列频率信号,然后再对这些频率信号进行倍频、分频和混频,最后得到大量的频率信号。其优点是频率稳定度高,频率转换时间短(可达 μs 量级),能做到任意小的频率间隔。缺点是系统中要用到大量的混频器、滤波器等,从而造成体积大,成本高,安装调试复杂,故只用于要求很高的场合。

间接频率合成技术是利用锁相技术来产生大量的具有高稳定度和高精度的频率源。由于间接频率合成器的关键部件是锁相环,故通常称为锁相环频率合成器。由于锁相环频率合成器的主要部件都易于集成,一般只加一个分频器和一个一阶低通滤波器,故其具有体积小、重量轻、成本低、安装和调试简单等优点。又由于锁相环频率合成器在性能上逐渐接近直接频率合成器,所以它在电子技术中得到了日益广泛的应用,并在应用中得到迅速发展。

本节将介绍应用广泛的数字式锁相环频率合成器,内容包括工作原理、实现方法等等。

7.1.1　通用锁相环频率合成器的系统结构

图 7-1 为一种最简单的频率合成器的结构框图。

锁相环电路是一个负反馈环路。它通常由基准频率源、鉴相器、低通滤波器、压控振荡器组成。

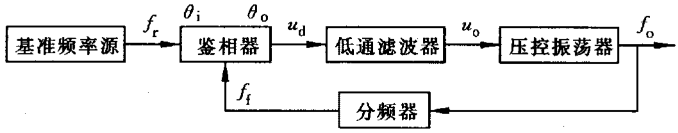


图 7-1 锁相环频率合成器结构

基准频率源提供一个频率稳定的信号，其频率为 f_r，一般用精度很高的石英晶体振荡器产生，是锁相环的输入信号。

鉴相器是一个误差检测元件。它比较基准频率源的输出信号相位（设为 θ_i）和压控振荡器输出信号相位（设为 θ_o），输出一个电压信号 u_d，其大小取决于两个输入信号的相位差。然后，u_d 通过低通滤波器滤除高频分量，输出控制电压 u_o 去控制压控振荡器。压控振荡器的输出信号频率 f_o 与控制电压 u_o 成一定比例，而分频器将锁相环的输出信号反馈给鉴相器，形成一个负反馈，从而使输入信号和输出信号之间的相位差保持恒定。

分频器为环路提供一种反馈机制，当 $N=1$ 时，锁相环系统的输出信号频率等于输入信号频率，即

$$f_o = f_f$$

信号锁定后有

$$f_o = f_f = f_r$$

当环路分频器 $N>1$，有

$$f_o = N \cdot f_f$$

环路锁定后有

$$f_f = f_r$$

$$f_o = N \cdot f_f = N \cdot f_r$$

若改变 N，则 $f_f \neq f_r$，环路失锁，这时环路就进行频率捕捉和相位捕捉。经过一段时间后，环路重新进入锁定状态，频率合成器完成一个频率转换过程，此时频率合成器输出为一个新的稳定频率。

当环路处于稳定状态时，输出和输入之间存在一定量相位误差。而对于输入信号频率和输出信号频率而言，二者却是成比例的，这时环路处于锁定状态，这是锁相环路的一个特点。用这种方法可以得到非常精确的频率控制。而其他的频率控制方法，在稳态时总是存在一定的频率误差。

7.1.2 通用锁相环频率合成器的主要技术指标

1. 频率间隔

锁相环频率合成器的频率间隔定义为输出频率 f_o 的最小频率间隔。当频率合成器中的分频器为整数分频时，频率合成器的频率间隔等于参考频率 f_r。

2. 频率范围

频率合成器的最大输出频率与最小输出频率之差定义为频率范围。频率范围也常用

频率覆盖系数 f_{max}/f_{min} 和相对频率带宽 $2(f_{max}-f_{min})/(f_{max}+f_{min})$ 表示，其中 f_{max} 和 f_{min} 分别为频率合成器的最大和最小输出频率。

3. 频率转换时间

频率合成器的输出从某一频率变到另一频率所需的最小时间称为频率转换时间。它包括频率转换时间和相位转换时间。如果不采取任何措施，锁相环频率合成器的频率转换时间一般不小于 10ms 量级。

4. 波道数(频率点数)

频率合成器可以输出的频率点的总数为频率合成器的波道数，或称频率点数。

5. 长期稳定度

频率合成器的输出频率的长期稳定度即为频率合成器的稳定程度。一般锁相环频率合成器的长期稳定度与参考信号源的长期稳定度相同，所以必须采用高精度的信号源。

6. 相位噪声

频率合成器内部各部件产生的相位噪声对频率合成器输出产生的影响的总称为频率合成器的相位噪声。相位噪声与频率短期稳定度等价。

图 7-1 所示的锁相环频率合成器只有一个反馈环路，统称为单环频率合成器。它是一种最简单的结构，实际应用中要复杂一些，例如基准源由晶体振荡器和参考分频器组成。

单环频率合成器的特点是电路结构简单，制作和调试方便，其频率间隔 f_r、频率范围主要取决于压控振荡器的输出频率-电压特性。但是单环频率合成器的缺点有

(1) 当分频器的分频比较大的时候，输出噪声也较大。

(2) 频率间隔不能做得太大(通常不能小于 1kHz)。

下面介绍的 MC68HC08 中的锁相环频率合成器就是一种单环频率合成器。

7.2 MC68HC08 时钟发生模块

7.2.1 MC68HC08 时钟发生模块概述

在 Motorola 的 MC68HC08 微控制器中，时钟发生模块主要负责产生与系统工作相关的几个时钟信号。MC68HC08 的时钟模块结构比较复杂，本章的介绍将主要从应用角度出发，对于一般读者，可直接参考本章最后所附的例程。

与 MC68HC05 不同，MC68HC08 的系统时钟不仅可以由外部晶振输入分频得到，还可以由内部锁相环频率合成器提供，如图 7-2 所示。锁相环频率合成器是 MC68HC08 系列微控制器中最具特色的模块之一，它可以在使用外接 32kHz 晶振的情况下，通过软件

编程得到最大 8MHz 的总线时钟频率。在以往不采用锁相环电路的微控制器应用系统中，晶振电路由于工作频率高(通常都有几兆赫至几十兆赫)，是一个很大的干扰源。在系统设计、线路板布局时都必须仔细考虑，采用各种方法降低对外界的干扰。而采用锁相环频率合成器作为微控制器时钟源后，由于可采用 32kHz 的晶振，比起通常使用的 2～8MHz 晶振，大大降低了对外辐射干扰，提高了系统的可靠性。这在同类的产品中具有无可比拟的优越性。同时由于锁相环频率合成器的输出频率可以用软件设定，这就为用户编程提供了更大的灵活性。

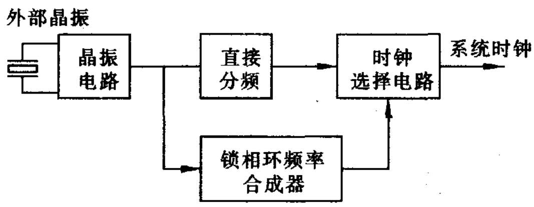


图 7-2 时钟发生模块结构示意

在微控制器时钟发生电路中采用锁相环电路是 Motorola 系列产品中首先引入的，本章将以较大篇幅介绍锁相环模块的原理、特点和编程方法，同时提供几个具体的应用实例。

7.2.2 MC68HC08 时钟发生模块的内部组成

1. 时钟发生模块的 3 个子模块

MC68HC08 中的时钟发生模块系统结构如图 7-3 所示。

时钟发生模块包括 3 个主要的子功能模块：

(1) 晶振电路。晶振电路通过外接石英或陶瓷振荡器产生稳定不变的时钟信号 CGMXCLK，CGMXCLK 直接输出给系统集成模块和 AD 转换器，同时也输出到时钟选择模块。CGMXCLK 经过缓冲后输出到锁相环频率合成器，作为信号源，这一路信号称为 CGMRCLK。

(2) 锁相环频率合成器。锁相环电路通过压控振荡器产生 CGMVCLK 信号，输出到时钟选择电路。其频率可通过软件编程控制。

(3) 时钟选择电路。时钟发生模块的输出信号 CGMOUT 有两种来源：直接采用晶振电路产生的 CGMXCLK 信号二分频和采用压控振荡器产生的 CGMVCLK 信号二分频。时钟选择电路可以通过软件编程决定采用哪种信号来源。

2. 时钟发生模块的输出信号和控制信号

(1) 输出信号

① CGMXCLK

CGMXCLK 是晶振电路的输出信号，频率等于晶振的频率。信号的精度和质量取决

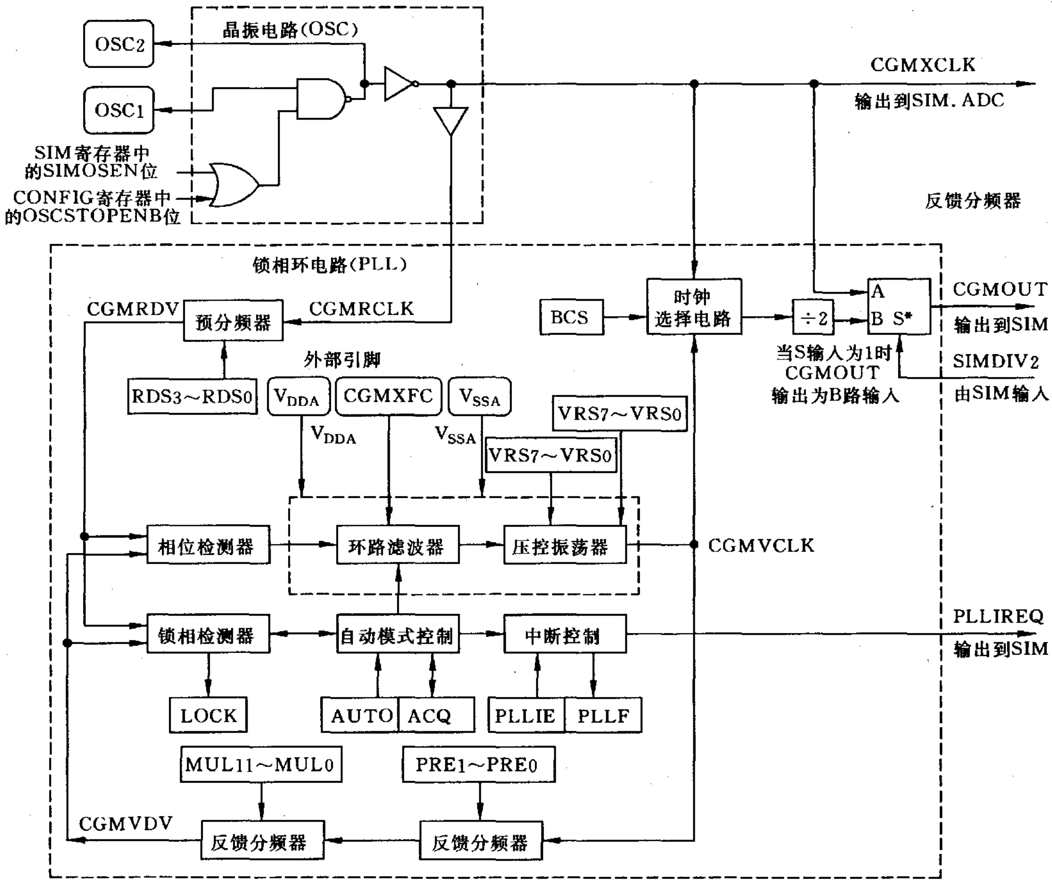


图 7-3　时钟发生模块系统结构

于外接晶振和外界因素，且在系统启动阶段，CGMXCLK 是不稳定的。

② CGMOUT

CGMOUT 是时钟发生模块的输出信号，该信号送入系统集成模块，系统集成模块产生微控制器的时钟信号。CGMOUT 占空比为 50%，经过 2 分频后产生总线时钟，CGMOUT 的来源可编程选定为晶振电路输出 CGMXCLK 的 2 分频或压控振荡器电路的输出 CGMVCLK 2 分频。

(2) 与时钟模块有关的控制信号

① SIMOSCEN

SIMOSCEN 来自系统集成模块，是 PLL 电路和晶振电路的使能信号。

② OSCSTOPENB

OSCSTOPENB 是 CONFIG 寄存器中和晶振相关的控制位(详见 CONFIG 寄存器的相关内容)。若置位，则晶振电路在 STOP 模式下可继续工作；若复位(缺省情况)，则晶振电路的行为受 SIMOSCEN 标志位控制，在 STOP 模式下将关闭晶振电路。控制信号与时钟发生模块的控制寄存器有关，详见控制寄存器有关章节。

3. 时钟发生模块的 CPU 中断信号 PLLIREQ

PLLIREQ 是由 PLL 电路锁相检测器产生的中断信号。

7.2.3 MC68HC08 时钟发生模块的外部引脚

在典型应用情况下，时钟发生模块需要 9 个外接器件，其中晶振电路中需要 5 个，PLL 电路需要 2 到 4 个，如图 7-4 所示。

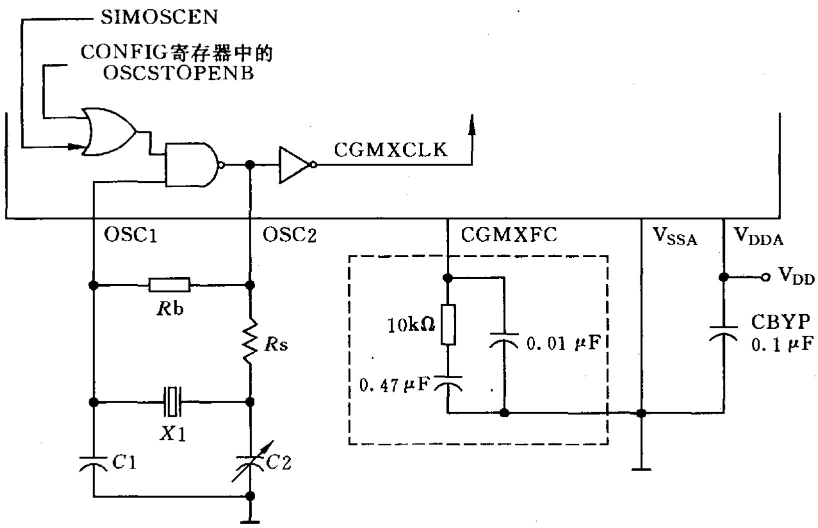


图 7-4 时钟发生电路的外部连接

1. 与时钟电路有关的引脚

MC68HC08 中和时钟部分相关的引脚有 OSC1，OSC2，CGMXFC，V_{SSA}，V_{DDA}。

(1) OSC1，时钟输入引脚。OSC1 将引脚的输入信号连至晶振电路的反向放大器。

(2) OSC2，时钟输出引脚。OSC2 引脚输出经过反向的输入信号。若采用外接信号源作为时钟输入，OSC2 引脚可以悬空，也可以连接到其他微控制器的 OSC1 输入引脚。

(3) CGMXFC，外部滤波电容器引脚。CGMXFC 是 PLL 电路环路滤波器所必需的，连接一个外接滤波网络。

(4) V_{DDA}，PLL 电路电源引脚。V_{DDA}是 PLL 电路的电源输入，应该和 V_{DD}引脚连接同一电源，在布线时 V_{DDA}引脚应该加滤波电容，同时用尽量短的引线连接。

(5) V_{SSA}，PLL 电路电源引脚。V_{SSA}为 PLL 电路的电源地线，和系统地线连接。

为了减小干扰，提高系统电磁兼容性，在元件布局上，滤波网络应该尽量靠近微控制器，用最短的连线连接，同时远离其他布线。

图 7-4 中晶振电路采用下述元件：晶体 *X*1、电容 *C*1 和 *C*2、反馈电阻 *R*b、串行电阻 *R*s。

$C1$ 和 $C2$ 的取值可参考晶振厂家给出的典型值，一般取 10～36pF，$C1$ 与 $C2$ 的值应该略有差异，以利于晶振电路起振。典型情况下，Rs 取 330kΩ，Rb 为 10MΩ。晶振采用 32.768kHz。

2. 与 PLL 电路有关的元件

PLL 电路采用下述元件：

(1) 跨接电容 CBYP。跨接电容用于稳定锁相环电源引脚 V_{DDA}，一般取 0.1μF 左右。

(2) 滤波网络。滤波网络为芯片内部的锁相环电路提供误差电平，连接到 CGMXFC 引脚。元件参数可参考图 7-4，如用户不打算在应用中使用锁相环电路部分，这一部分电路可以不接，让 CGMXFC 引脚悬空。

特别要提醒用户注意的是，时钟部分会对其他电路造成干扰，在电路板布线时应该将时钟电路的外接器件都用地线围绕起来。如果电路板空间允许，晶振应该采用卧式安装，将外壳焊接在地上，最大限度地减小对外界的电磁干扰。

7.2.4 MC68HC08 时钟发生模块的晶振电路

晶振电路由片内的反向放大器、整形电路及外接的晶体组成。OSC1 引脚为输入引脚，将信号引入片内的反向放大器，OSC2 为输出引脚，如图 7-5 所示。

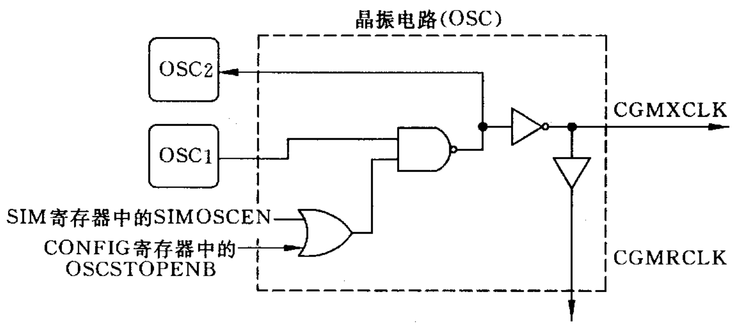


图 7-5 晶振电路结构

CGMXCLK 信号为晶振电路的输出信号，频率等于外接晶振的标称频率。CGMXCLK 信号经过缓冲后产生 CGMRCLK 信号，作为锁相环频率合成器的参考时钟。晶振电路有两个控制信号，分别是 SIM 寄存器中的 SIMOSCEN 位和 CONFIG 寄存器中的 OSCSTOPENB 位。这两个寄存器的设置请参考相关章节。

CGMXCLK 信号还可以作为微控制器中其他需要精确时钟信号的模块的输入。需要注意的是，由于外接晶振电路使用分立元件，元器件的分布参数及环境因素，如温度、湿度等都会对时钟信号的精度产生影响。若用户对时钟精度有很高要求，可以采用外接的高精度频率源作为微控制器系统的时钟输入，此时信号由微控制器 OSC1 引脚输入，OSC2 引脚可以悬空。

微控制器的晶振电路是一个强噪声源，由于工作频率高(通常都是几兆赫的频率)，很

容易对系统其他电路造成干扰，干扰信号主要是时钟信号的 3 倍频。晶振电路同时也容易被其他电路干扰，造成系统死机。

解决对其他电路干扰的方法包括在保证性能的前提下尽量降低时钟频率，在电路布局上缩短与晶振电路有关元器件的引脚等。目前市场上，同一档次的微控制器产品中，Motorola 系列的工作频率是最低的。例如，同样达到 1MHz 的内部时钟，Intel 系列需要 12MHz 的外部晶振，而 Motorola 系列只需要 4MHz。MC68HC08 中又引入了锁相环电路，只需要 32kHz 的晶振，这又进一步降低了系统对外干扰。

7.2.5 MC68HC08 时钟发生模块的锁相环频率合成器

MC68HC08 中的 PLL 电路是一种单环的锁相环频率发生电路。与 7.1 节介绍的有所不同的是，它有两种工作模式：获取模式和跟踪模式。这两种模式能提供不同的频率输出精度和稳定度，具体采用哪种模式要根据对输出信号的要求。两种模式之间的切换可以自动进行，也可以手动进行。

1. PLL 电路结构和信号关系

PLL 模块由以下电路组成：压控振荡器、参考时钟分频器、反馈时钟分频器、压控振荡器电路分频器、鉴相器、环路滤波器和锁相检测器。MC68HC08 中的 PLL 电路结构简图、信号以及和寄存器控制位的关系如图 7-6 所示。

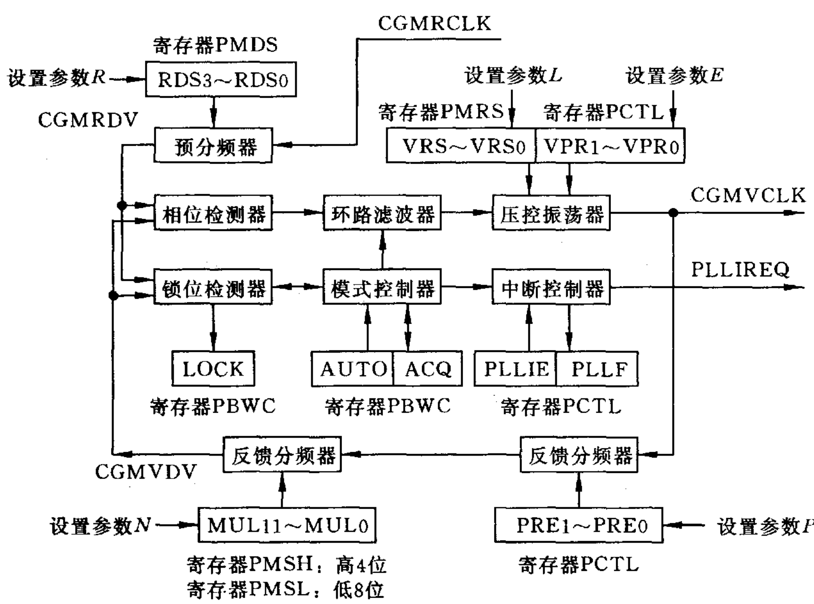


图 7-6 PLL 电路模块

图 7-6 中的 CGMRCLK 是晶振电路的输出信号，也是 PLL 电路的参考时钟输入，其频率 f_{RCLK} 通过可编程预分频电路输入 PLL，预分频器将 f_{RCLK} 除以分频因子 R。预分频

器的输出为最终参考时钟 CGMRDV，其频率 $f_{RDV}=f_{RCLK}/R$。R 因子由寄存器 PMDS 中的 RDS3～RDS0 位设置。

在 PLL 电路前设置预分频器主要是为了适应不同的外接晶振。只有限制输入频率在一定范围内，PLL 电路才能处于最佳的工作状态。例如，当外部晶振频率在 30kHz～100kHz 范围内时，通常设定 $R=1$；若外接更高频率的晶振，选择合适的 R 值，将输入信号频率调整到 30kHz～100kHz 的工作范围内。

压控振荡器电路的输出信号是 CGMVCLK，频率设为 f_{VCK}，将通过两个反馈分频器反馈到相位检测器，其中第一个反馈分频器分频因子为 P，第二个为 N，分别在寄存器 PMSH 和 PCTL 中设置。反馈分频器的输出为相位检测器的输入信号 CGMVDV，相位检测器比较反馈信号 CGMVDV 和参考时钟信号 CGMRDV，根据这两个信号之间的相位差产生一个修正电压，环路滤波电路通过 CGMXFC 引脚上外接电容稳定修正电压，修正电压的值将控制压控振荡器的输出信号 CGMVCLK 的频率。

锁相检测器比较压控振荡器电路的反馈信号 CGMVDV 和参考时钟信号 CGMRDV，判断 PLL 的模式和锁相环的工作状态。

2. PLL 电路的获取模式和跟踪模式

和所有的自动控制网络一样，PLL 电路的工作包括反馈和调整的过程。因此总是需要一个从不稳定到稳定的过程，尤其是启动过程和改变频率输出的过程，所以 MC68HC08 中为 PLL 电路设置了两种工作方式，以适应在不同条件下的要求。PLL 电路可以通过手动或自动方式调整为如下两种工作方式：

(1) 获取模式。在获取模式下，压控振荡器电路提供最大的频率稳定精度。此种模式适用于 PLL 电路的启动过程的不稳定状态，或有强烈的噪声、压控振荡器电路严重偏离了工作频率的情况。在获取模式下，PLL 带宽控制寄存器中的 ACQ 标志位清零。这也就是给用户程序一个信号，表明当前时钟信号的稳定程度。在程序中可以查询 ACQ 标志位，避免在时钟不稳定的时候进行高精度定时操作。

(2) 跟踪模式。在跟踪模式下，压控振荡器电路仅对输出信号频率做小范围的调整，此时电路响应速度最快，可以对相位变化快速响应，但同时噪声容限也降低了。一般情况下，当压控振荡器电路工作基本稳定后，PLL 电路自动进入跟踪模式。

PLL 电路的两种工作模式可以在程序中根据不同的使用条件设定，但对于一般用户，推荐使用自动方式。在自动方式下，系统会自动选择工作模式（关于自动方式，见 7.4 节）。

7.2.6 MC68HC08 时钟发生模块的时钟选择电路

这部分电路选择作为系统总线时钟源 CGMOUT 的输入信号，来源可以是晶振电路产生的 CGMXCLK 信号的 2 分频，或锁相环电路产生的 CGMVCLK 信号的 2 分频，时钟选择电路是一个晶体管二选一数据选择器。当从一种时钟源切换到另一种时钟源后，电路要等待 3 个时钟周期，目的是等待输入信号稳定，在此期间内，CGMOUT 停止。二选

一数据选择器的输出信号再经过 2 分频产生总线时钟。所以，系统总线时钟频率是 CGMOUT 的二分之一，是时钟信号源 CGMXCLK 或 CGMVCLK 的四分之一。

PLL 控制寄存器中的 BCS 位决定 CGMOUT 的信号来源。当 PLL 电路未工作时不能选压控振荡器信号作为信号源。同样，当已经选定压控振荡器作为信号源后，不能关闭 PLL 电路。操作的顺序也应当注意，PLL 电路的打开和关闭操作不能和压控振荡器电路的选定或取消操作同步进行，当 L 因子为 0 时，压控振荡器电路同样不能作为时钟源。若用户操作过程中出现上述错误，则晶振电路将被强制作为时钟源。

7.3 MC68HC908GP32 时钟发生模块

7.3.1 MC68HC908GP32 时钟发生模块的寄存器

与时钟发生模块相关的有以下寄存器：

(1) PLL 控制寄存器；

(2) PLL 带宽控制寄存器；

(3) PLL 倍频因子寄存器高位；

(4) PLL 倍频因子寄存器低位；

(5) PLL 压控振荡器范围选择寄存器；

(6) PLL 参考分频因子寄存器。

下面分别介绍各个寄存器的意义和设置方式。

1. PLL 控制寄存器

PLL 控制寄存器，简称 PCTL，为 8 位寄存器，包含 PLL 中断允许位和标志位、参考时钟选择位、预分频因子位、压控振荡器指数选择位。其地址为 $0036，如图 7-7 所示。

	位	7	6	5	4	3	2	1	0	
$0036		PLLIE	PLLF	PLLON	BCS	PRE1	PRE0	VPR1	VPR0	PCTL
	复位	0	0	1	0	0	0	0	0	

图 7-7 PLL 控制寄存器

(1) PLLIE——PLL 中断允许位

PLLIE 控制位可读写，控制当 PLL 带宽控制寄存器的 LOCK 状态位翻转时是否产生 CPU 中断。LOCK 位翻转时表示 PLL 电路的锁相状态发生变化，可能是进入稳定状态，也可能是有强烈的干扰(见关于 PLL 带宽控制寄存器的介绍)。如果用户程序对于系统时钟要求较高，应当在中断程序中判断锁相环是否处于稳定状态，并采取相应措施。

当 PLL 带宽控制寄存器中的 AUTO 控制位为 0 时，此时为锁相环的自动调整状态(参见 PLL 带宽控制寄存器的介绍)，用户程序不干预 PLL 电路的状态，PLLIE 为只读，且为零，表示不开中断。微控制器复位后清除 PLLIE 位。

1＝PLL 中断许可。

0＝PLL 中断禁止。

(2) PLLF——PLL 中断标志位

PLLF 为中断标志位，当 LOCK 状态位翻转时被置位。若此时 PLLIE 控制位为“1”，即允许中断，那么 PLLF 的置“1”便产生一次中断请求，读 PLL 控制寄存器将清除 PLLF 位。注意，在用户中断程序的末尾应当清除 PLLF 位。微控制器复位后清除 PLLF 位。

1＝LOCK 状态位改变。

0＝LOCK 状态位未改变。

注意，任何对 PLL 控制寄存器的读或改写操作都会清除 PLLF 位。

(3) PLLON——PLL 开关检测

此位为可读写位，用于启动 PLL 电路并激活压控振荡器时钟 CGMVCLK，当压控振荡器正作为基准时钟源时(BCS＝1)PLLON 不能被清零，要关闭 PLL 应该先不选 PLL 为时钟源(BCS＝0)，再清除 PLLON 位。微控制器上电复位后此位置位。

1＝PLL 打开。

0＝PLL 关闭。

(4) BCS——CGMC 基时钟选择位

此位为可读写位，用于选择 CGM 模块输出信号 CGMOUT 的输入信号源。当 PLLON 标志位为 0 时，BCS 不能置位。当 BCS 位改变后，随后的 3 个时钟周期为电路稳定时间。在此期间内 CGMOUT 无输出，RESET 清除 BCS 位。

1＝选择 PLL 电路为时钟源，CGMVCLK 2 分频后驱动 CGMOUT。

0＝选择晶振分频为时钟源，CGMXCLK 2 分频后驱动 CGMOUT。

注意，PLLON 和 BCS 控制位都有内部保护机制。当 PLL 关闭时，可以制止时钟选择电路选择压控振荡器电路作为时钟源。因此，操作顺序应该是：先打开 PLL 电路(PLLON 置 1)，再选择 PLL 电路为时钟源(BCS 置 1)。或者是先选择晶振分频为时钟源(BCS 置 0)，再关闭 PLL 电路(PLLON 置 1)。也就是说，若 PLL 电路处于关闭状态(PLLON＝0)，选择 CGMVCLK 需要 2 次写入 PCTL(先置位 PLLON，再置位 BCS)，而不能一次写入 PCTL。

(5) PRE1 和 PRE0——反馈分频器分频位

此位为可读写位，设置反馈分频器分频因子 P(详见 PLL 编程一节)，如表 7-1 所示。当置位 PLLON 位，打开 PLL 电路后，PRE1 和 PRE0 不能被写入。也就是说，对 PRE1 和 PRE0 的写操作必须在 PLL 关闭的情况下进行。RESET 清除 PRE1 和 PRE0 位。

(6) VPR1 和 VPR 0——压控振荡器选择位

这 2 位可读写，如表 7-2 所示，它设置压控振荡器模块的参数 E 和控制参考频率 f_{VRS}(详见 PLL 编程一节)。对 VPR1 和 VPR0 的写操作也必须在 PLL 关闭的情况下。当 PLLON 位为 1 时，VPR1 和 VPR0 不能被写。RESET 清除这 2 位。

注意，E 的值不能为 3(VPR1 和 VPR0 不能为 11)，否则电路工作不正常。

表 7-1　反馈反频器分频表

PRE1 和 PRE0	P	反馈反频器分频值 2^P
00	0	1
01	1	2
10	2	4
11	3	8

表 7-2　压控振荡器选择位表

VPR1 和 VPR0	E	压控振荡器选择值 2^E
00	0	1
01	1	2
10	2	4

2. PLL 带宽控制寄存器

PLL 带宽控制寄存器,简称 PBWC,如图 7-8 所示。该寄存器执行下列操作:

	位	7	6	5	4	3	2	1	0	
$0037		AUTO	LOCK	ACQ	0	0	0	0	R	PBWC
	复位	0	0	0	0	0	0	0	0	

图 7-8　PLL 带宽控制寄存器

(1) 自动或手动(软件控制)带宽模式;

(2) 指示 PLL 何时为锁定状态;

(3) 在自动带宽控制模式,指示 PLL 何时处于获取模式或跟踪模式;

(4) 在手动方式下,强制 PLL 进入获取模式或跟踪模式;

(5) R 为保留位。用户程序中不能使用。

(1) AUTO——自动带宽控制位

此位可读可写,选择自动或手动带宽模式。当采用手动模式时(AUTO=0),用户可以编程设定采用跟踪模式还是获取模式,可以增加软件的灵活性。当初始化 PLL 为手动方式时(AUTO=0),在打开 PLL 之前必须清除 ACQ 位。RESET 清除 AUTO 位。手动方式需要用户对时钟发生模块有相当的了解。对于普通用户,推荐使用自动方式。

1=自动方式。

0=手动方式。

(2) ACQ——获取模式状态位

当 AUTO 位为 1(设为自动方式),ACQ 为只读位,指示 PLL 电路目前是进入获取模式还是跟踪模式;当 AUTO 位为 0(设为手动方式),ACQ 为可读可写寄存器,控制 PLL 是否进入获取模式或跟踪模式。

在自动带宽控制模式(AUTO=1)时,手动控制方式下最后一次写入 ACQ 的值将被保存在临时空间,直到再次采用手动方式后执行写入。RESET 清除此位。

1=跟踪模式。

0=获取模式。

(3) LOCK——锁相指示位

当 AUTO 位为 1 时(设为自动方式),LOCK 为只读位,且当压控振荡器的时钟

CGMVCLK 完成锁定(工作在程序设定频率)后置 1,表示时钟稳定。当 AUTO 位为 0 (设为手动方式)时,LOCK 始终读出为 0,无意义。因为手动方式下,是由用户决定是采用跟踪模式或获取模式。在此位写入 1 是留给测试使用的,所以手动方式下这一位只能写入 0。RESET 将清除 LOCK 位。

1=压控振荡器频率已经锁定。

0=压控振荡器频率还未稳定。

由于手动和自动工作模式的选择是时钟发生模块应用中的一处难点,为了读者能更好地理解各个状态和 PLL 带宽控制寄存器控制位之间的关系,我们将手动、自动和中断状态下 PLL 带宽控制寄存器控制位状态总结如下:

① 自动方式下

在自动控制模式下(控制位 AUTO=1),锁相检测器将自动在获取模式与跟踪模式之间切换。在用户程序中,也可以在自动带宽控制模式下,通过软件检测当前处于何种模式,从而判断 CGMVCLK、CGMOUT 等时钟信号是否稳定可用。如果 PLL 中断允许,软件可以等待 PLL 中断,然后检查 LOCK 状态位。无论是获取模式还是跟踪模式,当 LOCK 被置位后,表示压控振荡器时钟信号已经稳定可靠了;而如果压控振荡器时钟被选为基准时钟,并且 LOCK 位清零,表明 PLL 电路正处于严重的噪声环境中,用户程序必须根据实际情况采取相应的补救措施。例如避免对实时性要求较高的操作。

当 PLL 电路处于自动带宽控制模式时,将出现下列情况:

- ACQ 为只读,指示环路滤波器的工作状态。ACQ 为 1 表示当前为跟踪模式,如果是此种模式,表明压控振荡器输出信号已经稳定在一定范围内,反之,ACQ=0 表明是获取模式,压控振荡器的信号还不稳定。
- LOCK 状态位为只读,指示 PLL 电路的锁相状态。LOCK 状态位为 1 表示压控振荡器输出信号已经稳定在一定范围内,反之,LOCK=0 表示输出信号还未锁定,或正处于调整状态。
- PLLIE=1 表示 CPU 中断允许,当 PLL 的锁相状态改变时产生中断,同时翻转 LOCK 位。

② 手动方式下

PLL 电路也可以工作在手动方式下(AUTO=0),手动方式适用于对时钟精度要求较高的应用系统。手动方式下将出现下列情况:

- ACQ 为可读写控制位,控制滤波器的工作模式。在打开手动模式之前,ACQ 必须为清零状态。
- 当设定 PLL 控制寄存器 PCTL 的控制位 PLLON=1,打开 PLL 后,软件必须等待一个给定时间 T_{ACQ},等待 PLL 工作稳定。
- 选择 PLL 电路输出为时钟源(BCS=1),进入跟踪模式后,软件必须等待一个给定时间 TAL,等待电路稳定。
- LOCK 位在此模式下不用。
- CGMC 的 CPU 中断被禁止。

③ PLL 产生的中断

当 PLL 带宽控制寄存器中的 AUTO=1 时，系统为自动调整模式。此时 LOCK 位可反映时钟的稳定程度。除了用查询 LOCK 位的方式外，每次 LOCK 位改变状态时，PLL 可以产生 CPU 中断请求。如果此时 PLL 控制寄存器中的 PLLIE 置位，将允许 CPU 的 PLL 中断。中断响应后，PCTL 中的中断标志位 PLLF 置位。但当 AUTO 位清零，系统为手动方式，PLL 中断禁止，此时 PLLF 读出为 0。

中断发生后，软件应该读取 LOCK 位，判断中断发生的原因，是进入锁相稳定状态还是由稳定状态变得不稳定。如果压控振荡器的时钟不稳定，软件应该采取相应的措施，例如终止对精度要求高的定时器操作等。采用中断方式比用软件查询 LOCK 位效率更高。如果用户程序不要求定时操作，或对于时钟频率稳定性要求并不高，可以关闭 PLL 中断，以避免频繁的中断处理降低系统性能，同时占用大量的系统堆栈空间。

3. PLL 反馈分频器高字节寄存器和低字节寄存器

PLL 反馈分频器高字节寄存器（简称 PMSH，见图 7-9）中的 4 位（MUL11～MUL8）和低字节寄存器（简称 PMSL，见图 7-10）中的 8 位（MUL7～MUL0）一起共同组成 12 位的有效位，记作 PMSH:PMSL，设置反馈系数 *N*（见 PLL 参数计算一节）。

	位	7	6	5	4	3	2	1	0	
$0038		0	0	0	0	MUL11	MUL10	MUL9	MUL8	PMSH
	复位	0	0	0	0	0	0	0	0	

图 7-9 PLL 反馈分频器高字节寄存器

	位	7	6	5	4	3	2	1	0	
$0039		MUL7	MUL6	MUL5	MUL4	MUL3	MUL2	MUL1	MUL0	PMSL
	复位	0	1	0	0	0	0	0	0	

图 7-10 PLL 反馈分频器低字节寄存器

MUL11～MUL0 这 12 位为可读可写位，选择压控振荡器电路的反馈模块的分频因子 N（详见 PLL 电路编程一节）。由于分频系数不能为 0，如果将 PMSH:PMSL 设置为 $0000，硬件将默认为 $0001，避免出错。RESET 初始化 PMSH:PMSL 为 $0040，即缺省的分频因子为十进制的 64。

注意：反馈分频器寄存器有内部的保护机制，当 PLLON=1 时，此位不能被写。即 PLL 电路工作后不能改变 MUL11～MUL0 的值。对 MUL11～MUL0 的写入操作应当在 PLL 电路关闭的情况下进行。

PMSH7～PMSH4 为未定义位，总为 0。

4. PLL 压控振荡器范围选择寄存器

PLL 压控振荡器范围选择寄存器简称 PMRS，在其他 MC68HC08 的技术文档中可能被称为 PVRS。

PMRS 的功能是对压控振荡器电路进行设置，如图 7-11 所示。

位	7	6	5	4	3	2	1	0	
$003A	VRS7	VRS6	VRS5	VRS4	VRS3	VRS2	VRS1	VRS0	PMRS
复位	0	1	0	0	0	0	0	0	

图 7-11 PLL 压控振荡器范围选择寄存器

VRS7～VRS0 为可读可写位，控制 PLL 电路中心频率线性因子 L（详见 PLL 电路编程一节）。当 PCTL 中的 PLLON 控制位为 1 时（打开 PLL），VRS7～VRS0 不能被写。也就是说对 VRS7～VRS0 的写操作只能在 PLL 关闭时进行。在 VRS7～VRS0 中写入 $00 将禁止 PLL 电路，并清除 PLL 控制寄存器 PCTL 中 BCS 控制位。RESET 初始化寄存器内容为 $40，即设定范围为十进制的 64。

注意：压控振荡器范围选择寄存器有内部保护机制。当 PLL 电路打开时，PLLON＝1，寄存器为写保护。PLL 压控振荡器范围选择寄存器必须初始化正确，否则 PLL 电路不能正确完成锁相。

5. PLL 预分频因子寄存器

PLL 预分频因子寄存器简称 PMDS，在 MC68HC08 的其他技术文档中被称为 PROS。

PMDS 的功能是设置预分频器，如图 7-12 所示。

位	7	6	5	4	3	2	1	0	
$003B	0	0	0	0	RDS3	RDS2	RDS1	RDS0	PMDS
复位	0	0	0	0	0	0	0	1	

图 7-12 PLL 参考分频因子寄存器

RDS3～RDS0 为预分频因子设置位，可读可写，设置参考时钟预分频因子 R（详见 PLL 电路编程一节）。当 PLLON 位为 1 时，RDS3～RDS0 不可写。对 RDS3～RDS0 的操作要在 PLL 电路关闭的情况下进行。由于 R 不能为 0，所以如果在 RDS3～RDS0 写入 0，硬件将默认为 $01 以避免出错。RESET 设置 RDS0 为 1。

注意：PMDS 在内部有保护机制。当 PLL 打开时 PLLON＝1，寄存器为写保护。

PMDS7～PMDS4 这 4 位未定义。

7.3.2 MC68HC908GP32 时钟发生模块中锁相环电路参数计算

下面介绍对锁相环电路的各种参数计算及软件编程，可满足各种频率晶振条件下的寄存器设置，但作为一般用户，采用手册推荐的 32kHz 晶振和典型参数值，可不必关心参数的计算过程，直接跳过本节，阅读编程示例。

下面介绍计算锁相环参数的方法。以下公式中的变量若有小数，则四舍五入到整数。

1. PLL 电路参数计算

（1）选择希望得到的总线时钟 f_{BUSDES}。

(2) 计算希望得到的压控振荡器时钟(4 倍于总线时钟)。

$$f_{VCLKDES} = 4 \times f_{BUSDES} \tag{7-1}$$

(3) 选择 PLL 参考时钟频率 f_{RCLK} 以及预分频因子 R。在典型情况下,外接晶振为 32.768kHz,此时取 $R=1$。

频率误差将以比率 f_{RCLK}/R 进行修正,为了提高频率的稳定性和提高反馈响应速度,频率和修正比率应当越大越好,压控振荡器的频率必须是此比率的整数倍。压控振荡器输出频率 f_{VCLK} 和参考时钟 f_{RCLK} 之间的关系为

$$f_{VCLK} = \frac{2^P N}{R}(f_{RCLK}) \tag{7-2}$$

在应用设计许可的噪声容限下,f_{RCLK} 的取值应该尽可能的大,当然也要考虑其他模块对时钟信号的要求,例如由 CGMXCLK 提供时钟的模块。选择预分频因子 $R=1$,P 与 N 确定后,由式(7-2)可确定总线工作时钟。

如果对系统的噪声容限要求较高,选择 f_{RCLK} 为 f_{BUSDES} 的整数除数,$R=1$,如果 f_{RCLK} 不能满足要求,可以参考式(7-3),由实际选定的 f_{RCLK} 推导出 R,最后确定能导出最小 R 值的 f_{RCLK}。

$$R = \text{round}\left[R_{max} \times \left\{\left(\frac{f_{VCLKDES}}{f_{RCLK}}\right] - \text{integer}\left[\frac{f_{VCLKDES}}{f_{RCLK}}\right)\right\}\right] \tag{7-3}$$

(4) 选择压控振荡器分频因子

$$N = \text{round}\left[\frac{R \times f_{VCLKDES}}{f_{RCLK}}\right] \tag{7-4}$$

减小 N/R 得到最小的 R。

(5) 由于在 MC68HC08 中用 12 位表示 N,所以 N 的最大值 N_{max} 为 \$FFF。如果得到的 N 小于 N_{max},取 $P=0$;如果 N 大于 N_{max},P 的取值可参考表 7-3。然后重新计算

$$N = \text{round}\left[\frac{R \times f_{VCLKDES}}{f_{RCLK} \times 2^P}\right] \tag{7-5}$$

表 7-3 P 与 N 的关系

当前的 N 值	P	当前的 N 值	P
$0 < N \leqslant N_{max}$	0	$N_{max} \times 2 < N \leqslant N_{max} \times 4$	2
$N_{max} < N \leqslant N_{max} \times 2$	1	$N_{max} \times 4 < N \leqslant N_{max} \times 8$	3

(6) 计算并检验压控振荡器频率 f_{VCLK} 和总线频率 f_{BUS}。

$$f_{VCLK} = (2^P \times N/R) \times f_{RCLK} \tag{7-6}$$

$$f_{BUS} = (f_{VCLK})/4 \tag{7-7}$$

(7) 选择压控振荡器的指数因子 E,可根据表 7-4 选择。

(8) 计算并检验压控振荡器的中心频率 f_{VRS},中心频率是 PLL 电路能够达到的最小和最大频率的中点。

$$f_{VRS} = (L \times 2^E) f_{NOM} \tag{7-8}$$

$$|f_{VRS} - f_{VCLK}| \leqslant \frac{f_{NOM} \times 2^E}{2} \tag{7-9}$$

表 7-4 f_{VCLK}与 E 的关系

频率/Hz	E
$0 < f_{VCLK} < 9830400$	0
$9830400 \leqslant f_{VCLK} < 19660800$	1
$19660800 \leqslant f_{VCLK} < 39321600$	2

(9) 验证以上选择的因子 P、R、N、E 和 L，这可以通过比较 f_{VCLK}、f_{VRS}和 $f_{VCLKDES}$得出。正常情况下，f_{VCLK}必须处于 $f_{VCLKDES}$的噪声容限内，并且 f_{VRS}必须尽量接近 f_{VCLK}。注意：超过推荐的最大总线时钟频率或压控振荡器频率将很可能损坏微控制器。

在用以上介绍的方法进行 PLL 电路的编程时，可能会遇到一些特殊情况，如 R、N 的计算结果为 0，L 对于所引入的方程无意义等。下面说明意外情况的处理方法。

若计算结果 R、N 为零，则一律取 1。

若 L 的值为 0，则 PLL 电路不工作。

(10) 程序中的操作步骤。

① 在 PLL 控制寄存器 PCTL 的 PRE 位中设定分频因子 P；

② 在 PLL 控制寄存器 PCTL 的 VPR 位中设定分频因子 E；

③ 在 PLL 分频寄存器 PMSL 和 PMSH 中设定 N；

④ 在 PLL 压控振荡器选择寄存器 PMRS 中设定参数 L；

⑤ 在参考时钟寄存器 PMDS 中设定 R。

2. PLL 电路参数计算应用举例

例 7.1 用 2MHz 的外接晶振产生 4MHz 的总线频率。

(1) 选择总线时钟频率 $f_{BUSDES} = 4MHz$。

(2) 计算压控振荡器频率 $f_{VCLKDES} = 4 \times f_{BUSDES} = 16MHz$。

(3) 选择 PLL 参考频率 $f_{RCLK} = 2MHz$。取 $R=1$。

(4) 选择压控振荡器分频因子 N。

$$N = f_{VCLKDES}/f_{RCLK} = 16MHz/2MHz = 8$$

由于 N 小于 N_{max}，取 $P=0$。

(5) 计算并检验压控振荡器时钟和总线时钟。

$$f_{VCLK} = N \times f_{RCLK}$$

$$f_{VCLKDES} = N \times f_{RCLK} = 8 \times 2MHz = 16MHz$$

$$f_{BUSDES} = f_{VCLKDES}/4$$

$0 < f_{VCLK} < 9,830,400$，可取 $E=0$。

(6) 选择压控振荡器线形因子 L。

$$L = f_{VCLK}/f_{NOM} = 16MHz/4.9152MHz = 3.255 = 3$$

其中 $f_{NOM} = 4.9152MHz$。

(7) 计算并检验压控振荡器的中心频率 f_{VRS}。

$$f_{VRS} = L \times f_{NOM} = 3 \times 4.9152\text{MHz} = 14.75\text{MHz}$$

由计算可知，f_{VRS}在适合范围 2.5MHz～32MHz 内。

表 7-5 中给出了一些常用的总线频率和参数的数值。这些总线频率通常是标准波特率的整数倍。

表 7-5　总线频率与参数值表

f_{BUS}/MHz	f_{RCLK}/kHz	R	N	P	E	L
2.0	32.768	1	F5	0	0	D1
2.4576	32.768	1	12C	0	1	80
2.5	32.768	1	132	0	1	83
4.0	32.768	1	1E9	0	1	D1
4.9152	32.768	1	258	0	2	80
5.0	32.768	1	263	0	2	82
7.3728	32.768	1	384	0	2	C0
8.0	32.768	1	3D1	0	2	D0

7.3.3 MC68HC908GP32 时钟发生模块编程举例

本章所列举的例程都是在外接 32.768kHz 晶振的情况下。

定义时钟发生模块寄存器地址的头文件

```
******************************************************************************
*      时钟寄存器头文件
*
******************************************************************************
    PCTL          EQU        $0036
    B_PLLON       EQU        5
    B_BCS         EQU        4
    PBWC          EQU        $0037
    B_AUTO        EQU        7
    B_LOCK        EQU        6
    PMSH          EQU        $0038
    PMSL          EQU        $0039
    PMRS          EQU        $003a
    PMDS          EQU        $003b
```

例 7.2　将系统 PLL 初始化为 2.4576MHz。

```
******************************************************************************
*      PLL 初始化,设置总线频率为 2.4576MHz
```

```
*
*******************************************************************************
    MIRR_PMSH    DB    $01
    MIRR_PMSL    DB    $2c
    MIRR_PMRS    DB    $80
    //其他程序段
    CLR      PCTL               ;设置 PCTL,关闭中断
    MOV      #$01,PCTL          ;设置预分频器因子 R 为 1
    LDA      MIRR_PMSH          ;设置反馈分频器高位(参数 N)
    STA      PMSH
    LDA      MIRR_PMSL          ;设置反馈分频器低位(参数 N)
    STA      PMSL
    LDA      MIRR_PMRS          ;设定 L 因子
    STA      PMRS
    BSET     B_PLLON,PCTL       ;启动压控振荡器时钟
    BSET     B_AUTO, PBWC       ;设置工作模式自动
    BSET     B_BCS, PCTL        ;选择 PLL 信号为系统时钟源
```

例 7.3 将中断时钟设置为 8MHz。

```
*******************************************************************************
*    PLL 初始化,设置总线频率为 8MHz     *
*******************************************************************************
    MIRR_PMSH    DB    $03
    MIRR_PMSL    DB    $D1
    MIRR_PMRS    DB    $D0
    //其他程序段
    CLR      PCTL               ;设置 PCTL,关闭中断
    MOV      #$01,PCTL          ;设置预分频器因子 R 为 1
    LDA      MIRR_PMSH          ;设置反馈分频器高位(参数 N)
    STA      PMSH
    LDA      MIRR_PMSL          ;设置反馈分频器低位(参数 N)
    STA      PMSL
    LDA      MIRR_PMRS          ;设定 L 因子
    STA      PMRS
    BSET     B_PLLON,PCTL       ;启动压控振荡器时钟
    BSET     B_AUTO, PBWC       ;设置工作模式自动
    BSET     B_BCS, PCTL        ;选择 PLL 信号为系统时钟源
```

7.3.4 MC68HC908GP32 时钟发生模块在 WAIT、STOP 指令中的状态

WAIT 指令不影响 CGM 的状态，在进入 WAIT 模式之前，软件可以通过清除 PLL

控制寄存器中的BCS位和PLLON位关闭PLL电路，从而降低功耗。对功耗要求不高的系统可以在WAIT指令期间保持PLL电路的工作，这样当系统脱离WAIT模式时，可以在最短时间内进入正常工作方式。

在STOP方式下，如果系统配置寄存器中的OSCSTOPENB位被清除了(缺省)，STOP指令将关闭CGM(晶振电路和锁相环电路)，并将时钟电路的输出信号CGMXCLK和CGMOUT设置为低。

如果在STOP运行时，系统时钟CGMOUT是由压控振荡器时钟CGMVCLK的2分频得到的，执行STOP指令后，PLL自动清除PLL控制寄存器PCTL中的BCS位，从而选择晶振时钟CGMXCL的2分频当作CGMOUT。当系统脱离STOP模式后，BCS位保持清零状态。也就是说，PLL电路不能从STOP中自动恢复，需要程序重新设置。

第 8 章　输入输出 I/O

8.1　并行 I/O

8.1.1　并行口

MC68HC08 中的并行口都是双向并行口。如 MC68HC908GP32 有 A,B,C,D,E 5 个并行口,其中 A,B,D 口为 8 位,当作为输入时可软件设定为内部上拉。C 口为 7 位,E 口为 2 位。由于封装引脚的限制,除了 C 口为单一功能的双向 I/O 口外,其余并口都与其他功能模块复用。MC68HC908GP32 中的 A 口全部可作为键盘中断输入引脚,B 口可定义为 8 路 A/D 转换器的模拟量输入引脚,D 口与 SPI 和定时器输入捕捉、输出比较引脚公用,E 口与 SCI 口公用。

表 8-1　MC68HC908GP32 中的 I/O 引脚功能

并口	位	方向寄存器	功能复用		引脚
A	0	DDRA0	KBD	KBIE0	PTA0/KBD0
	1	DDRA1		KBIE1	PTA1/KBD1
	2	DDRA2		KBIE2	PTA2/KBD2
	3	DDRA3		KBIE3	PTA3/KBD3
	4	DDRA4		KBIE4	PTA4/KBD4
	5	DDRA5		KBIE5	PTA5/KBD5
	6	DDRA6		KBIE6	PTA6/KBD6
	7	DDRA7		KBIE7	PTA7/KBD7
B	0	DDRB0	ADC	CH0	PTB0/ATD0
	1	DDRB1		CH1	PTB1/ATD1
	2	DDRB2		CH2	PTB2/ATD2
	3	DDRB3		CH3	PTB3/ATD3
	4	DDRB4		CH4	PTB4/ATD4
	5	DDRB5		CH5	PTB5/ATD5
	6	DDRB6		CH6	PTB6/ATD6
	7	DDRB7		CH7	PTB7/ATD7

续表

并口	位	方向寄存器	功能复用		引脚
C	0	DDRC0			PTC0
	1	DDRC1			PTC1
	2	DDRC2			PTC2
	3	DDRC3			PTC3
	4	DDRC4			PTC4
	5	DDRC5			PTC5
	6	DDRC6			PTC6
D	0	DDRD0	SPI		PTD0/$\overline{SS}$
	1	DDRD1			PTD1/MISO
	2	DDRD2			PTD2/MOSI
	3	DDRD3			PTD3/SPSCK
	4	DDRD4	TIM1		PTD4/T1CH0
	5	DDRD5			PTD5/T1CH1
	6	DDRD6	TIM2		PTD6/T2CH0
	7	DDRD7			PTD7/T2CH1
E	0	DDRE0	SCI		PTE0/TxD
	1	DDRE1			PTE1/RxD

不同的 MC68HC908GP32 芯片封装包含不同数量的 I/O 引脚，44 脚的 QFP 封装包含 33 条 I/O 引脚(见表 8-1)，42 脚的 SDIP 较之 QFP 减少了 PTC5 和 PTC6，40 脚的 PDIP 又减少了 PTD6 和 PTD7。

MC68HC908GP32 的并行 I/O 的功能和输出引脚如表 8-1 所示(QFP 封装)。

MC68HC908GP32 中的双向并行口的结构可参看图 8-1。

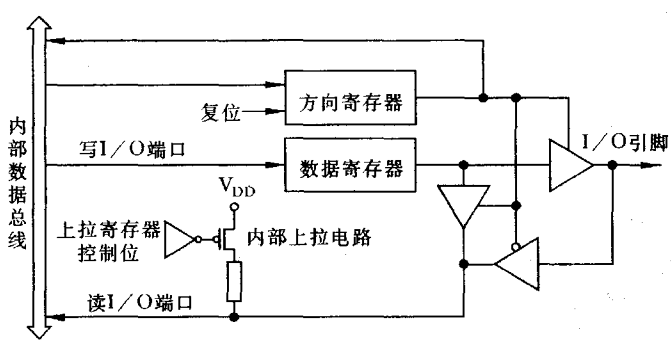


图 8-1　并行 I/O 口的结构

其中，并行口每一引脚的方向（输入或输出）由对应的数据方向寄存器的对应位决定。例如 A 口，若想把 PA3 作为输入引脚，只需将 A 口的数据方向寄存器 DDR 的第 3 位置 0 即可；同样，若想把 PB7 作为输出引脚，只需将 B 口的 DDR 寄存器的第 7 位置 1 即可。

A 口、C 口和 D 口均具有内部上拉电阻，并各有一个上拉控制寄存器。例如 A 口为 PTAPUE，上拉寄存器置 1，设置相应的 I/O 引脚上拉，如图 8-2 所示。C 口、D 口的上拉控制寄存器分别为 PTCPUE 和 PTDPUE。

	位	7	6	5	4	3	2	1	0	
$000D		PTAPUE7	PTAPUE6	PTAPUE5	PTAPUE4	PTAPUE3	PTAPUE2	PTAPUE1	PTAPUE0	PTAPUE
	复位	0	0	0	0	0	0	0	0	

图 8-2　端口 A 上拉控制寄存器

8.1.2　并行口应用举例

用并行口实现多个 LED 数码管的动态显示，电路图如图 8-3 所示，用 B 口输出显示字符，用 C 口选通某一个数码管。数码管采用共阳极数码管，每个数码管用一个 PNP 晶体管控制选通。当 C 口某一位为 0 时晶体管导通，显示 B 口输出的字符。

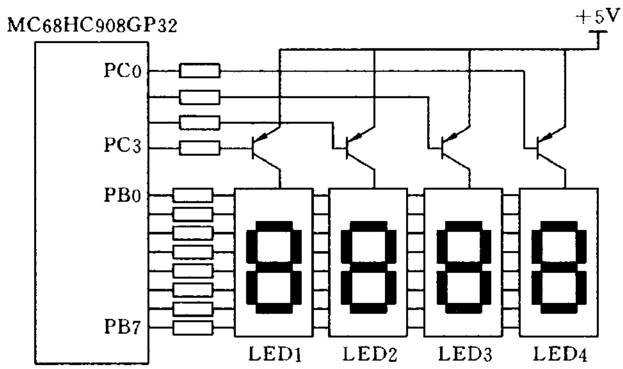


图 8-3　LED 驱动电路

在软件中用 PC0～PC3 选择数码管，当某一管脚为低电平时相应的数码管被选中。在定时器中断程序中轮流点亮 4 个 LED 数码管，只要刷新频率大于 50Hz，人的眼睛就看不出数码管的闪烁。在程序中，要显示的字符放在 LED 缓冲区中，由定时器中断轮流获取，输出到 B 口。NUM 为当前要显示的数码管索引，取值为 0～3。LEDTABLE 为数码管选通表项，例如要点亮 LED1，则需要向 C 口输出数值 0xF7（LED1 由 PC3 选通）。其他数码管的选通表项可以依次求出。程序中可以由索引值 NUM 查表取得要显示的字符和选通表项。采用查表的方法可以大大简化选通的操作，减小中断程序运行时间。

程序中所用到的全局变量：

```
LED        RMB    4    ;显示字符缓冲区
NUM        RMB    1    ;显示字符索引
```

```
LEDTABLE      FDB $F7FB,$FDFE        ;数码管选通表
```

初始化程序如下：

```
……
MOV           #$0xFF,PORTB          ;初始化B口和C口的输出锁存器,
                                    ;以免输出不定状态
MOV           #$0xFF,PORTC          ;
MOV           #$0xFF,DDRB           ;B口和C口都是输出口
MOV           #$0xFF,DDRC
MOV           #0,NUM                ;初始化显示字符索引
……
```

显示刷新程序(定时器中断服务程序)：

```
    ……
    MOV       #$0xFF,PORTC          ;消隐LED显示
    LDX       NUM                   ;取得当前要显示的数码管索引
    LDA       LED,X                 ;取得当前要显示的字符
    STA       PORTB                 ;显示字符
    LDA       LEDTABLE,X            ;取得选通表项
    STA       PORTC                 ;选通数码管,开显示
    INCX                            ;字符索引加1
    STX       NUM                   ;保存修改过的NUM
    CPX       #4                    ;如果为4,则NUM清零
    BNE       NOTADD
    MOV       #0,NUM
NOTADD:
    ……                              ;其他程序段
    RTS                             ;中断返回
```

8.2 异步串行通信接口SCI

MC68HC08是MC68HC05系列微控制器的增强型。与MC68HC05系列微控制器的SCI系统相比,MC68HC08系列微控制器的SCI系统功能上更加强大。如MC68HC08GP32具有硬件奇偶校验、噪声检测等功能。

MC68HC08系列微控制器的SCI系统是一个通用异步接收器/发送器·UART类型的异步通信接口,通过串行通信协议(如RS-232、RS-485等协议)同主机系统(如个人计算机、终端等)通信。加驱动后SCI能在相对长的距离上通信,多个微控制器可以通过各自的SCI相连组成串行通信网络。

在一般应用中,CPU简单地把数据写入并行数据寄存器即可实现发送一个格式化的串行字符,SCI系统完成发送数据的所有细节工作,包括附加起始位和停止位以符合串行

格式。发送器有一个两字节的发送队列(俗称双缓冲),这样 CPU 就有更多的时间准备其他字节了。SCI 的接收器自动探测一个字节的起始位,并通过采样接收数据。接收串行数据并变换成并行数据的所有工作均由 SCI 完成,不需要 CPU 的干预。接收到数据后,CPU 简单地从数据接收寄存器读取数据即可。

在这一节里,将首先简单介绍一下异步串行通信的基本概念,然后以 MC68HC908GP32 为例,详细介绍 MC68HC08 系列微控制器的 SCI 系统。对于 MC68HC08 系列中不带 SCI 系统的微控制器,如 MC68HC08JL3 等,将给出一个软件串口的例子。

8.2.1 RS-232C 标准

计算机数据通信主要采用异步串行通信方式。目前有很多串行通信标准可供选择,例如 RS-232、RS-485、20mA 电流环等。RS-232C 是美国电子工业协会正式公布的异步串行通信标准,也是目前最常用的串行通信标准,用来实现计算机与计算机之间、计算机与外设之间的数据通信。RS-232C 适用于设备之间的通信距离不大于 15m,传输速率最大为 20kB/s。

1. 接口信号

现在常用的 RS-232C 标准接口采用 9 芯插头座。具体的 RS-232C 串行标准接口信号定义请见表 8-2。

表 8-2 RS-232C 接口信号

引脚号	缩写符	信号方向	说　　明
1	NC/G		金属外壳保护地
2	RXD	输入	接收数据
3	TXD	输出	发送数据
4	DTR	输出	数据终端准备好
5	GND		信号地
6	DSR	输入	数据设置好
7	NC		空
8	CTR	输入	清接受缓冲区
9	NC		空

2. 电气特性

RS-232C 采用负逻辑,即逻辑 1 取－3V～－15V,逻辑 0 取＋3V～＋15V。RS-232C 接口的其他电气特性请参见有关文献。

3. RS-232C 连接方法

用 RS-232C 连接系统时,有近程方式和远程方式之分。近程方式是指通信距离小于

15m 的通信;远程方式指 15m 以上的长距离通信。本章只考虑近程方式。一般来讲,实现近程通信,只要用 RS-232C 电缆直接相连即可,但是要考虑电平匹配问题。

图 8-4 是一种常用的连接方式,不使用握手线,即不用握手线做数据流控制。也就是说,不控制数据流,或用软件方法,如 XON、XOFF 协议来控制数据流。

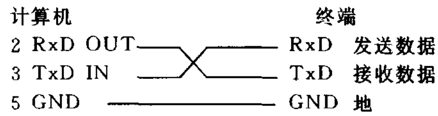


图 8-4 计算机/终端的完整连接

4. RS-232C 标准接口的实现及电平转换

前面已介绍,能完成异步通信的硬件电路称为 UART。符合 RS-232C 标准的 UART 很多,例如 PC 机用以实现 RS-232C 通信接口的 INS8250。完整的 UART 具有可编程性,每个字符可以包含 5 个或 6、7、8 个位,有 1 个或多于 1 个停止位及奇偶校验位。

RS-232C 的电平与大多数系统采用的 TTL/CMOS 电平不兼容,因此需要有接口电路完成电平转换工作。市场上有很多这种类型的芯片,如 Motorola 的 MC145407, Maxim 的 MAX232 等。为了与外部电路隔离,还可以采用光电隔离办法。此外市场上还有一些有源或无源的长线驱动模块,可以使传输距离提高到 2～10km。

8.2.2 标准不归零数据格式

SCI 使用标准的不归零数据格式(一个起始位,8 个或 9 个数据位和一个停止位),最常用的数据格式是 8 位的,关于数据格式参见图 8-5。

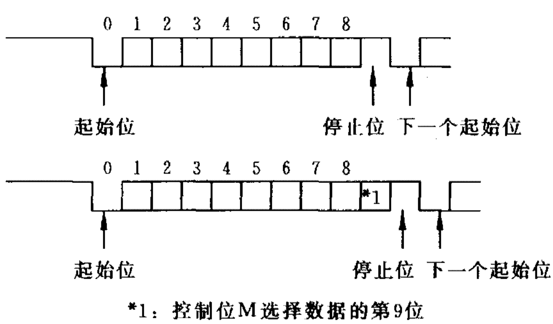


图 8-5 不归零数据格式

图 8-5 的上方是常见的 8 位数据格式;下方是 9 位数据格式。SCC1 寄存器中的控制位 M 设置成 1 时选择 9 位数据格式。不归零数据格式的基本特点为

(1) 高电平为逻辑 1,低电平为逻辑 0;

(2) 发送/接收数据空闲时 TxD、RxD 线为高电平;

(3) 发送、接收数据串的第一位是起始位(逻辑 0);

(4) 数据的 LSB 被首先发送、接收；

(5) 数据串的最后一位(第 10 位或第 11 位)是停止位(逻辑 1)；

(6) 当第 10 位或第 11 位为低时则中断(如果中断允许)。

8.2.3 SCI 功能

SCI 的功能结构框图如图 8-6 所示。

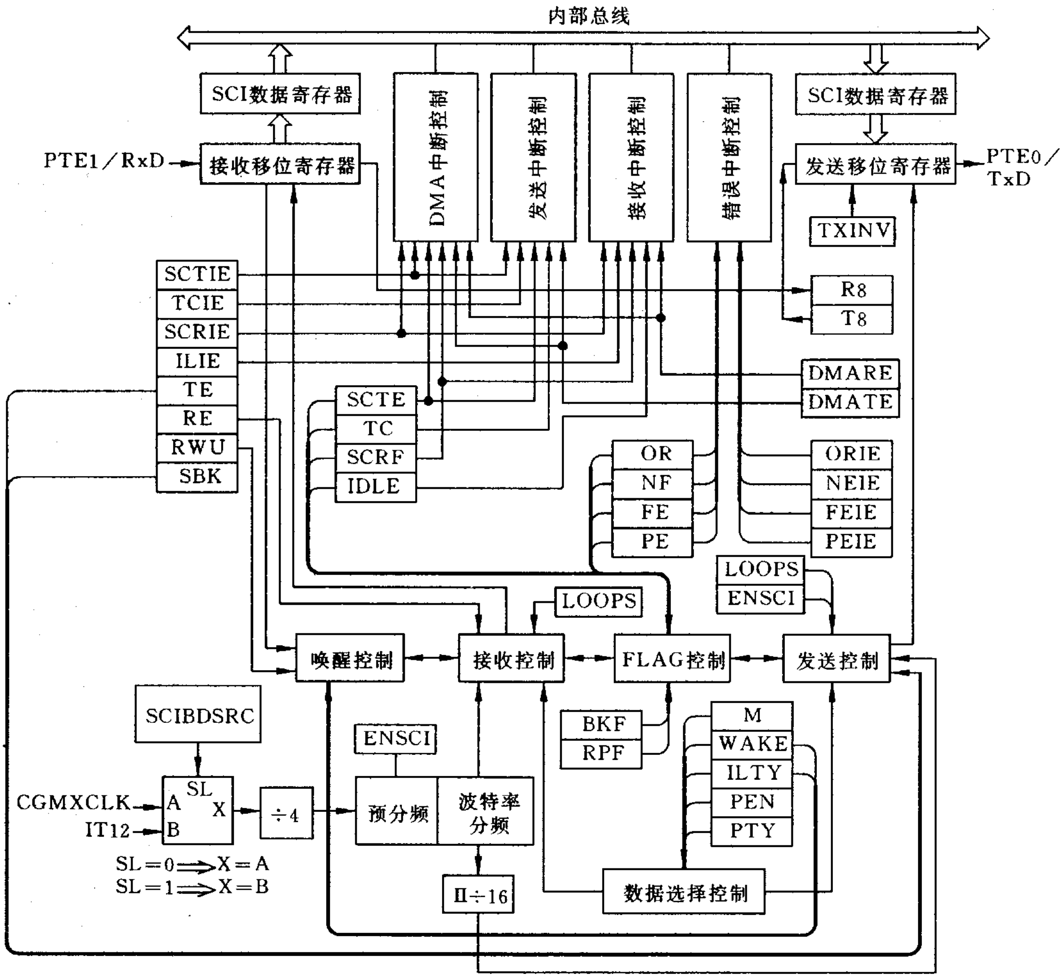


图 8-6 SCI 功能结构框图

微控制器芯片内的波特率发生器从微控制器振荡器经过分频得到各种标准波特频率。SCI 发送器和接收器功能独立，但使用同样的数据格式和波特率。MC68HC08 系列微控制器中也有一些产品的发送与接收的波特率不同。下面先简单介绍 SCI 的功能，后面再作详细说明。

1. SCI 系统功能

(1) 标准不归零格式；
(2) 全双工操作，发送器和接收器分别有独立的允许位；
(3) 软件预置 32 种波特率之一；
(4) 软件选择数据字长(每字 8 或 9 个位)；
(5) SCI 可用中断驱动，有 4 个分立的中断条件。

2. SCI 接收器功能

(1) 接收器唤醒功能(空闲线或地址位)；
(2) 空闲线探测标志；
(3) 错误检测标志——帧错误、噪声、溢出检测；
(4) 接收数据寄存器满标志。

3. SCI 发送器功能

(1) 发送数据寄存器空标志；
(2) 发送完成标志；
(3) 在发送过程中 CPU 随时可以中止发送行为。

8.2.4 SCI 发送器

软件写入 SCI 数据寄存器 SCDR 之后，数据送到发送缓冲区，移位寄存器从缓冲区得到数据。移位寄存器得到数据之后，即在它的 LSB 位装入 0 作为起始位，在最后一位装入 1 作为停止位。当串行控制寄存器 SCC2 中的 TE 位(发送允许)由 0 变 1 时，移位寄存器将全部装入 1，包括通常装入 0 的起始位，作为数据头。而在发送中止命令时，移位寄存器全部装入 0，包括通常装入 1 的停止位。发送缓冲区的附加位(第 9 位)由 SCI 控制寄存器 3 即 SCC3 的 T8 位提供。只有当 SCC1 的 M 位被设置为 1，选择 9 位数据格式时，第 9 位才有意义。M 位也控制空闲码和中止码的长度。

发送逻辑自动设置 SCI 状态寄存器 SCS1 中的发送数据寄存器空 SCTE 和发送结束 TC 状态标志。软件能随时读取这两个标志位。当数据从数据寄存器 SCDR 传送到移位寄存器以后，SCTE 标志置位，允许下一数据写入。当数据从移位寄存器向 TxD 引脚发送完毕时，TC 标志置位，这一标志位可以用于 Modem 控制。由此看来，SCI 发送器其实具有两个字节的缓冲区。若发送中断允许 SCTIE 和发送结束中断允许 TCIE 置位，SCTE 和 TC 标志将分别发出中断请求。

8.2.5 SCI 接收器

SCI 从 RxD 引脚接收数据，经过缓冲后驱动数据恢复模块。数据恢复模块实际上是

一个工作在16倍波特率下的高速移位寄存器。主移位寄存器的时钟频率就是波特率。数据恢复模块以波特率的16倍频率进行高速采样，完成诸如发现起始位、空闲线探测、噪声探测、仲裁逻辑之类的工作，向主接收移位寄存器提供每一位的值，同时也提供噪声标志状态指示。接收器在16次采样中取其中的7、8、9位或8、9、10位做3取2的多数占优的表决逻辑以决定每位的逻辑值。

SCI接收器的核心是接收串行移位寄存器，受SCI控制寄存器SCC2中接收允许RE位的控制。根据M位的不同，寄存器是10位或11位的。接收到停止位之后，数据由移位寄存器传送到SCDR数据寄存器，同时将接收数据寄存器满状态标志置位。当一个字符准备传送到接收缓冲区却发现上一个字符还未被取走时，就会发生溢出。此时数据不再传送，溢出OR状态标志置位以指出错误。

SCI接收器有3个中断源。这些标志可以用软件查询，或在允许时发出中断请求。设置接收中断允许SCRIE控制位就可以允许SCRF和OR状态标志发出硬件中断请求。设置空闲线中断允许ILIE控制位可以允许IDLE状态标志发出中断请求。

MC68HC08系列微处理器的SCI接收器具有唤醒功能。这个功能在多处理器系统中非常有用，暂时没有工作的微处理器可以处于休眠状态降低功耗，需要工作时再由主机将其唤醒。下面简单介绍唤醒功能。

在一个典型的多处理器系统中，软件协议通常在消息的开始判断地址。唤醒功能允许无关的微控制器忽略消息的剩余部分，同时禁止这些微控制器接收器的标志（和中断）处理，直到数据线返回到空闲状态。SCI接收器可以被一串包含10个（或11个）1的空闲字符串唤醒。发送器软件必须在发送连续消息的间隙提供必须的空闲字符串，并保证空闲字符串不在消息中出现。

还有一种将休眠中的微控制器唤醒的办法，即使微控制器接收到的字符的最高位为1。

8.2.6 SCI寄存器

SCI系统包括7个寄存器（SCC1，SCC2，SCC3，SCS1，SCS2，SCDR和SCBR）和2个外引脚（RxD和TxD）。当SCI接收器和发送器被允许时，SCI逻辑控制TxD和RxD，否则它们只用作普通的并行输入线。

下面详细介绍这些寄存器的主要功能。

1. 波特率寄存器

波特率寄存器（SCBR）见图8-7，用来选择SCI系统的波特率。波特率由微控制器总线时钟分频而来，发送器和接收器具有相同的波特率和数据格式。

	位	7	6	5	4	3	2	1	0	
\$0019		—	—	SCP1	SCP0	—	SCR2	SCR1	SCR0	SCBR
	复位	0	0	0	0	0	0	0	0	

图8-7 波特率寄存器

SCP1～SCP0 为预分频选择，预分频值见表 8-3。

表 8-3　预分频值

SCP1	SCP0	预分频值(PD)
0	0	1
0	1	3
1	0	4
1	1	13

SCR2～SCR0 为分频选择，分频值见表 8-4。

表 8-4　分频值

SCR2	SCR1	SCR0	分频值(BD)
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

波特率发生器分频原理见图 8-8。由此可见，对于一个给定的晶振频率，预分频和分频共 5 位可以提供 4×8=32 种波特率组合。

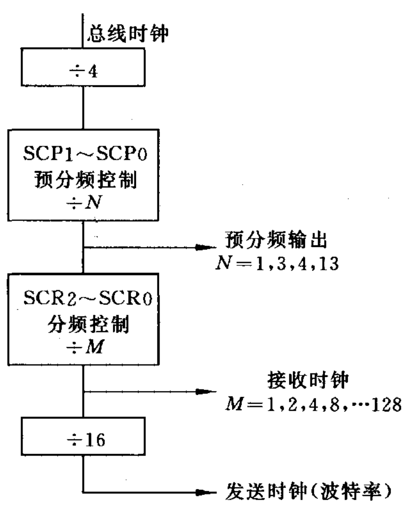


图 8-8　波特率发生器分频示意图

波特率计算公式如下：

$$波特率=内部总线时钟/(64\times PD\times BD)$$

其中，PD 为预分频值，见表 8-3；BD 为分频值，见表 8-4。

2. 串行通信控制寄存器 1

串行通信控制寄存器 1(SCC1)如图 8-9 所示。图中：

	位	7	6	5	4	3	2	1	0	
$0013		LOOPS	ENSCI	TXINV	M	WAKE	ILTY	PEN	PTY	SCC1
	复位值	0	0	0	0	0	0	0	0	

图 8-9 串行通信控制寄存器 1

(1) LOOPS 为循环模式选择位。当它置位时，选择循环模式，只有当发送和接收同时允许时，才能使用循环模式。在循环模式下，PTE1/RxD 引脚与 SCI 系统断开，发送器的输出直接接入接收器的输入。

(2) ENSCI 为 SCI 允许位。当该位置位时，允许 SCI。

(3) TXINV 为输出反相选择位。当该位置位时，发送输出为反码。

(4) M 为字符长度选择。它置位时为 9 位，清零时为 8 位。

(5) WAKE 为唤醒条件选择。

1＝地址唤醒；

0＝空闲线唤醒。

(6) ILTY 为空闲线方式选择。

1＝从停止位开始计数；

0＝从起始位开始计数。

(7) PEN 为奇偶校验允许位。置位时允许奇偶校验。

(8) PTY 为校验类型选择。

1＝奇校验；

0＝偶校验。

3. 串行通信控制寄存器 2

串行通信寄存器 2(SCC2)如图 8-10 所示，它是 SCI 子系统的主控制寄存器。这个寄存器可以允许和禁止发送器或接收器，允许和禁止系统中断、唤醒功能和送中止码功能。TIE、TCIE、RIE 和 ILIE 位是局部中断控制，决定 SCI 系统是查询状态标志还是发出硬件中断请求。

	位	7	6	5	4	3	2	1	0	
$0014		SCTIE	TCIE	SCRIE	ILIE	TE	RE	RWU	SBK	SCC2
	复位	0	0	0	0	0	0	0	0	

图 8-10 串行通信控制寄存器 2

图 8-10 中：

(1) SCTIE 为 SCI 发送中断允许位。置位时，允许 SCI 发送中断。

(2) TCIE 为发送完成中断允许位。置位时，允许发送完成中断。

(3) SCRIE 为接受中断允许位。置位时，允许 SCI 接收中断。

(4) ILIE 为空闲中断允许位。置位时，允许空闲中断。

(5) TE 为发送允许位。置位时，允许发送器。

(6) RE 为接收允许位。置位时，允许接收器。

(7) RWU 为接收唤醒位。置位时，接收器处于等待状态，并关闭接收中断。当 SCC1 的 WAKE 位决定的唤醒条件发生时，接收器退出等待状态，并清零 RWU 位。

(8) SBK 为终止位。置位时，发送终止码。

在一般系统中，TE 和 RE 写入 1 以允许 SCI 子系统的接收器和发送器工作。而 ILIE、RWU 和 SBK 不常用，写入 0 以禁止这些功能。如果不准备采用中断处理 SCI 工作，应向 TIE、TCIE 和 RIE 写入 0，否则要写入 1。例如，一个系统不用中断，就要在初始化时向 SCC2 装入 $0C。

4. 串行通信控制寄存器 3

串行通信控制寄存器 3(SCC3)如图 8-11 所示。图中：

	位	7	6	5	4	3	2	1	0	
$0015		R8	T8	DMARE	DMATE	ORIE	NEIE	FEIE	PEIE	SCC3
	复位	—	—	0	0	0	0	0	0	

图 8-11　串行通信控制寄存器 3

(1) R8 为接收位 8，只读。当选择 9 位字符时，该位为接收的最后一位。

(2) T8 为发送位 8。当选择 9 位字符时，该位为发送的最后一位。

(3) DMARE 和 DMATE 为保留位，应清零。

(4) ORIE 为接收器溢出中断允许位。置位时，允许中断。

(5) NEIE 为接收器噪声错中断允许位。置位时，允许中断。

(6) FEIE 为接收器帧错误中断允许位。置位时，允许中断。

(7) PEIE 为接收器奇偶错中断允许位。置位时，允许中断。

5. 串行通信状态寄存器 1

串行通信状态寄存器 1(SCS1)如图 8-12 所示，SCTE 标志位和 SCRF 标志位比较常用，而 TC 位和 IDLE 位一般不用。OR、FE 和 NF 位是否用来作为监控，应根据系统类型的不同要求决定。

	位	7	6	5	4	3	2	1	0	
$0016		SCTE	TC	SCRF	IDLE	OR	NF	FE	PE	SCS1
	复位	—	—	0	0	0	0	0	0	

图 8-12　串行通信状态寄存器 1

SCI 通信的连接方式主要有两种。其中之一是直接连接，例如微控制器和 PC 机之间的连接。在这种方式下，OR、NF 和 FE 之类的错误未必会真发生，因此典型做法是忽略掉。第二种类型是两个远地设备通过 Modem 连接。在这种连接下，很有可能发生错误，所以有必要在二者之间采用某种协议，允许在发生错误时再发送一遍数据。

图 8-12 中：

(1) SCTE 为发送缓冲区空标志。读出 SCS1 再写入 SCDR 时清 0。

(2) TC 为发送完成标志。

(3) SCRF 为接收缓冲区满标志。读出 SCS1 再读出 SCDR 时清 0。

(4) IDLE 为接收器空闲标志。读出 SCS1 再读出 SCDR 时清 0。

(5) OR 为接收器溢出标志。读出 SCS1 再读出 SCDR 时清 0。

(6) NF 为接收器噪声标志。在一位的 3 次测试中不相同时置位，读出 SCS1 再读出 SCDR 时清 0。

(7) FE 为接收器帧错误标志。读出 SCS1 再读出 SCDR 时清 0。

(8) PE 为接收器奇偶错标志。读出 SCS1 再读出 SCDR 时清 0。

6. 串行通信状态寄存器 2

串行通信状态寄存器 2(SCS2)如图 8-13 所示。图中：

	位	7	6	5	4	3	2	1	0	
$0017		—	—	—	—	—	—	BKF	RPF	SCS2
	复位	0	0	0	0	0	0	0	0	

图 8-13　串行通信状态寄存器 2

(1) BKF 为终止码标志位。置位时，表示检测到终止码，读出 SCS1 再读出 SCDR 时清 0。

(2) RPF 为正在接收标志。置位时表示正在接收。

7. 串行通信数据寄存器

串行通信数据寄存器(SCDR)如图 8-14 所示。它具有双重功能：读时是接收寄存器；写时是发送寄存器。

	位	7	6	5	4	3	2	1	0	
$0018		—	—	—	—	—	—			SCDR

图 8-14　串行通信数据寄存器

MC68HC08 系列微控制器的 SCI 子系统采用了发送器和接收器双缓冲技术，这是比 Intel 系列微控制器技高一筹的地方。SCI 的硬件缓冲区有两个字节，串行通信数据寄存器可以存储一个字节，移位寄存器可以存储一个字节。即使 CPU 因处理单个字符而延迟回应，也可以很容易地处理一串连续字符，因此用户在编制串行通信软件时有很大的回

旋余地。

下面用图 8-15 来说明一下双缓冲结构。

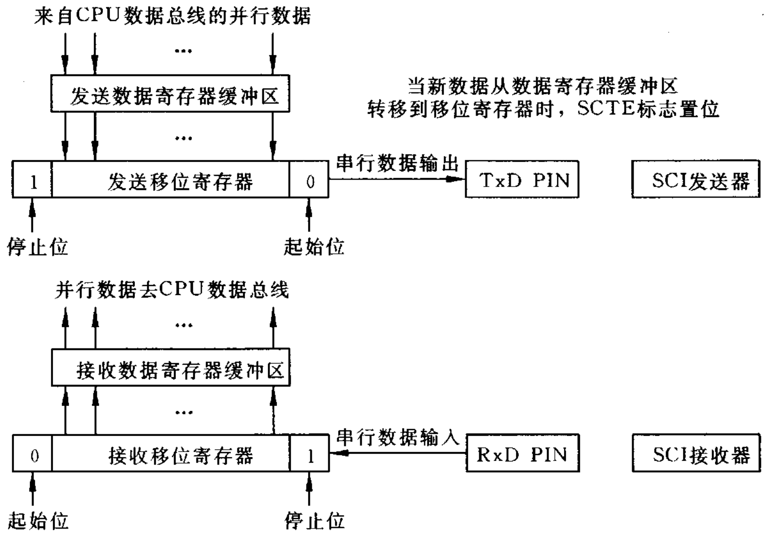


图 8-15　双缓冲技术图解

8.2.7 SCI 应用

SCI 系统在 MC68HC08 系列微控制器的应用中相当重要，是有力的开发手段和与外界联系的通道。下面叙述一下应用中的硬件开发和编制软件的一般流程。

1. 硬件环境

异步串行通信标准很多，可采用最简单也是最传统的 RS-232C 标准实现点对点的通信，即微控制器和 PC 机通信。在实际应用中可能需要一点对多点的通信，可以采用 RS-485 标准，使用一个主机、一对双铰线和多个从机微控制器通信，每个微控制器有自己的地址。主机发送消息到双铰线上，先送地址码，从机利用唤醒功能来识别地址，如果是自己的地址，则唤醒微控制器与主机通信，否则不予理睬。

2. 软件工作

SCI 系统的工作方式、使用方法都很灵活，对于一种还不太了解的芯片，有必要从最简单的使用方式入手。采用 8 位数据、无校验、一个停止位、4800 波特率，用查询方式发送、接收数据。一般来讲，查询方式简单易懂但浪费 CPU 的工作时间，而中断方式则效率高但比较复杂，为简单计，我们采用查询方式 。读者可以结合中断一章的内容，自己练习编写一个 SCI 中断程序。

(1) 初始化过程

① 写波特率寄存器设置波特率；

② 写控制寄存器 1(ENSCI, M,WAKE,PEN)设置数据长度并选择唤醒方法；

③ 写控制寄存器 2(SCTIE,TCIE,SCRIE,ILIE,TE,RE,RWU,SBK)允许期望的中断源,打开发送器和接收器,RWU 和 SBK 在初始化时应该写入 0。

④ 写控制寄存器 3(ORIE,NEIE,FEIE,PEIE)允许期望的出错中断。

下面列出了中断允许位及其控制的中断源的参考表。

表 8-5　中断允许位及其控制的中断源

允许位	标志位	中断源
SCTIE	SCTE	发送数据寄存器空
TCIE	TC	发送结束
SCRIE	SCRF,OR	接收数据寄存器满或溢出
ILIE	IDLE	探测到线空闲

(2) 查询方式发送流程

查询方式发送流程如图 8-16 所示。

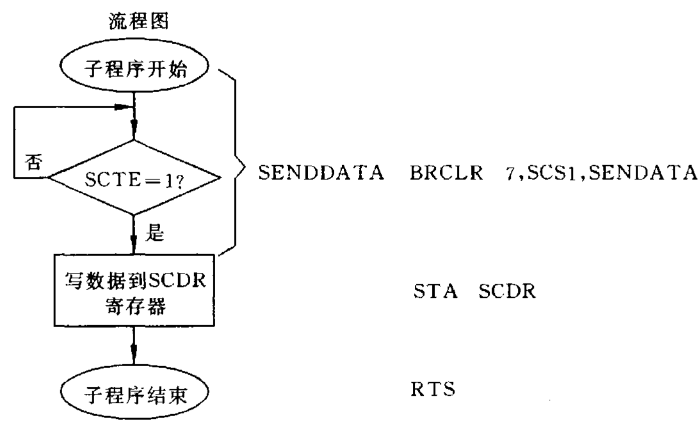


图 8-16　SCI 查询方式发送流程图

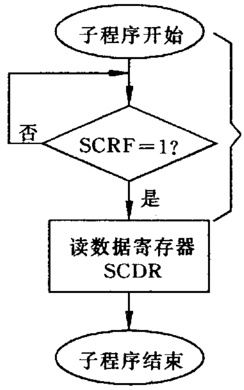
(3) 查询方式接收流程

查询方式接收流程如图 8-17 所示。

(4) SCI 应用例子

下面给出一个完整的 SCI 查询方式例子。微控制器与终端或 PC 机通过 RS-232 接口通信。用户通过键盘向微控制器发送一个 ASCII 字符,微控制器接收到这个字符后将其拆成两个 ASCII 字符,然后向终端发送回车(CR)、换行(LF)和“ $ ”字符,然后发送前面拆出的两个字符。发送结束后程序回到起点,等待终端发来的下一字符。拆字符的规则如下:第一个字节的低 4 位为接收到字符的高 4 位,高 4 位为 0011B;第二个字节的低 4 位为接收到字符的低 4 位,高 4 位为 0011B。例如,字符“A”的 ASCII 码为 $ 41,将其拆为 $ 34 和 $ 31,即 ASCII 码“4”和“1”。源程序如下所示。

流程图　　　　　　程序清单




```
GETDATA   BRCLR  5,SCS1,GETDATA

          LDA  SCDR

          RTS
```

图 8-17　SCI 查询方式接收流程图

```
****************************************
*   一个简单的 68HC08 SCI 程序范例      *
****************************************
SCBR      EQU $0019          ;定义寄存器内容
SCC1      EQU $0013
SCC2      EQU $0014
SCC3      EQU $0015
SCS1      EQU $0016
SCS2      EQU $0017
SCDR      EQU $0018
TEMP      EQU $40            ;定义一个字节的临时存储空间
TEMPHI    EQU $41            ;字节高 4 位对应的 ASCII 码存储空间
TEMPLO    EQU $42            ;字节低 4 位对应的 ASCII 码存储空间

          ORG $8000          ;定义程序起始地址
INITIAL:  LDA #%00100010     ;初始化开始
          STA SCBR           ;波特率定义为 4800(总线频率为 4.9152MHz)
          LDA #%01000000     ;设置 SCC1 寄存器内容
          STA SCC1
          LDA #%00001100     ;设置 SCC2 寄存器内容
          STA SCC2
          LDA #%00000000
          STA SCC3
START:    JSR GETDATA        ;接受一个字符
          STA TEMP           ;将其存储于临时空间
          AND #$0F           ;获取字节低 4 位
          ORA #$30           ;将低 4 位转化成 ASCII 码
          CMP #$39
          BLS ARN1           ;如果 ASCII 码在 30～39 之间则转移
```

```
          ADD # $7                    ;与偏移量相加
ARN1:     STA TEMPLO                  ;存储于 TEMPLO
          LDA TEMP
          LSRA                        ;右移 4 位
          LSRA
          LSRA
          LSRA
          AND # $0F                   ;获取字节高 4 位
          ORA # $30                   ;将高 4 位转化成 ASCII 码
          CMP # $39
          BLS ARN2
          ADD # $7
ARN2:     STA TEMPHI                  ;将其存储于 TEMPHI
          LDA # $0D
          BSR SENDATA                 ;发送回车符<CR>
          LDA # $0A
          BSR SENDATA                 ;发送换行符<LF>
          LDA #'$'
          BSR SENDATA                 ;发送字符"$"
          LDA TEMPHI
          BSR SENDATA                 ;发送高 4 位 ASCII 码
          LDA TEMPLO
          BSR SENDATA                 ;发送低 4 位 ASCII 码
          BRA START                   ;转移到 START
;获取一个字符,并将其存储于 A 寄存器
GETDATA:BRCLR 5,SCS1,GETDATA          ;SCRF 寄存器内容是否为 1
          LDA SCDR                    ;若是,则获取字符
          RTS                         ;返回
                                      ;发送一个字符
SENDATA:BRCLR 7,SCS1,SENDATA          ;SCTE 寄存器内容是否为 1
          STA SCDR                    ;若是,则发送字符
          RTS                         ;返回
```

8.2.8 软件串行口

大多数 MC68HC08 系列微控制器都具有硬件 SCI 系统,但是也有一些产品没有,如 MC68HC08JL3。对于这些产品的用户来讲这是一个缺憾。而且在很多实际应用中,需要更多的串行口,例如在多机系统中,一个用于多机通信,另外一个用于与本机控制台通信,所以在不增加硬件的条件下,充分利用微控制器自身的资源做一个软件串行口很有意义。这里将给出用 2 个普通的输入输出脚实现的软件串行口,其微控制器总线时钟为 1MHz,波特率为 4800,1 个起始位,8 个数据位,无校验位,1 个停止位。

软件串行口的发送和接收时序应该像真正的 SCI 一样，空闲状态为高电平，起始位为低电平，然后是 LSB，依次到 MSB，最后是停止位，停止位为高电平，具体时序请参见 8.2.2 节。在稍后提供的程序中共有 3 个模块：演示程序；发送函数；接收函数。下面依次进行说明。

1. 演示程序

这里编写了一个简单的主程序，用以演示如何调用发送、接收函数。在这段程序里，等待终端发送过来一个字符，然后再发送回终端，这样在终端上每发送出一个字符应该接收到同样的字符。

2. 发送函数

直接逐位发送，位间的延迟时间为 208μs。

3. 接收函数

接收函数循环检测起始位的发生，一旦检测到起始位，就按一定的时间间隔逐位采样（这里的时间间隔是 208μs）。为了提高抗干扰能力，每位采样 3 次。

程序清单如下：

```
* -------------------------------------------------------------- *
* 软件串行口
* 使用通用输入输出口
* 波特率=4800
* 微控制器总线频率为 1MHz
*  -------------------------------------------------------------- *
PORTC        EQU     $0002
DDRC         EQU     $0006
TX           EQU     $2
RX           EQU     $3
COUNT        EQU     $40
CHAR         EQU     $41
OUTCHAR      EQU     $42
*这是一个微控制器的 SCI 与终端通信的演示程序。
*MCU 将会接受一个由终端发送的 ASCII 码字符，然后将该 ASCII 字符码发送回终端。

        ORG        $0800
MAIN:
        BSET       TX,PORTC
        BSET       TX,DDRC
        BCLR       RX,DDRC
LOOP:
        JSR        GETCHAR                ;调用接受函数
```

```
        LDA     CHAR
        STA     OUTCHAR
        JSR     PUTCHAR             ;调用发送函数
        BRA     LOOP
*****************************************************
*描述:该模块将接受终端发送的一个 ASCII 码字符
*函数名:GETCHAR
*无输入变量
*输出变量:CHAR
*调用子函数:DELAY170,DELAY300
*该函数由主函数 MAIN 调用
*****************************************************
GETCHAR:
        BRSET   RX,PORTC,GETCHAR
        JSR     DELAY300
        LDA     #$08
        STA     COUNT
GETLP1:
        CLRA                        ;清 A 寄存器,用于与采样位相加
        BRSET   RX,PORTC,SAMP1      ;采样 3 次
SAMP1:  ADC     #$00
        BRSET   RX,PORTC,SAMP2
SAMP2:  ADC     #$00
        BRSET   RX,PORTC,SAMP3
SAMP3:  ADC     #$00
        LSRA
        LSRA
RX2 :
        ROR     CHAR
        JSR     DELAY170
        DEC     COUNT
        BNE     GETLP1
        CLRA
        BRSET   RX,PORTC,SAMP21     ;检查停止位
SAMP21:
        ADC     #$0
        BRSET   RX,PORTC,SAMP22
SAMP22:
        ADC     #$00
        BRSET   RX,PORTC,SAMP23
SAMP23:
        ADC     #$00
```

```
        LSRA
        LSRA
        BCC       GET_ERROR
        RTS
GET_ERROR:
        RTS
*******************************************************
* 描述:该模块将向终端发送一个 ASCII 码字符
* 函数名: PUTCHAR
* 输入变量:OUTCHAR
* 无输出变量
* 调用子函数:DELAY170
* 该函数由主函数 MAIN 调用
*******************************************************
PUTCHAR:
        SEI
        BCLR      TX,PORTC              ;起始位
        LDA       #$09
        STA       COUNT
PUT_LP1:
        JSR       DELAY170
        DEC       COUNT
        BEQ       PUT_STOP
        ROR       OUTCHAR
        BCC       PUT_TX2
        BSET      TX, PORTC
        BRA       PUT_LP1
PUT_TX2:
        BCLR      TX,PORTC
        BRA       PUT_LP1
PUT_STOP:
        NOP
        DECA
        DECA
        BSET      TX,PORTC
        JSR       DELAY170
        NOP
        NOP
        CLI
        RTS
*******************************************************
```

```
* 描述:该模块用于延时 300μs
* 函数名: DELAY300
* 无输入变量
* 无输出变量
* 由 GETCHAR 和 PUTCHAR 调用
*********************************************************
DELAY300:
        LDA     #$1D            ;2 个周期
DY_LP1:
        DECA                    ;1 个周期
        NOP                     ;1 个周期
        NOP                     ;1 个周期
        NOP                     ;1 个周期
        NOP                     ;1 个周期
        NOP                     ;1 个周期
        NOP                     ;1 个周期
        BNE     DY_LP1          ;3 个周期
        RTS                     ;4 个周期
*********************************************************
*********************************************************
* 描述:该模块用于延时 170μs
* 函数名: DELAY170
* 无输入变量
* 无输出变量
* 由 GETCHAR 和 PUTCHAR 调用
*********************************************************
DELAY170:
        LDA     #$10            ;2 个周期
DY_LP1:
        DECA                    ;1 个周期
        NOP                     ;1 个周期
        NOP                     ;1 个周期
        NOP                     ;1 个周期
        NOP                     ;1 个周期
        NOP                     ;1 个周期
        NOP                     ;1 个周期
        BNE     DY_LP1          ;3 个周期
        RTS                     ;4 个周期
*********************************************************
```

8.3 同步串行外设接口 SPI

大多数 MC68HC08 系列微控制器内部有一个同步串行外部设备通信接口，即 SPI

(serial peripheral interface),SPI 系统用于同标准外设芯片通信。这类芯片很多,例如串转并和并转串移位寄存器、A/D 变换器、LCD 控制器等等。微控制器还可以通过 SPI 组成一个通信速率比 SCI 高的同步网络,在一个小型系统中交换数据,完成较复杂的工作。

SPI 系统的工作原理好像一个分布式 16 位移位寄存器,一半在微控制器里(即 SPI),另外一半在外设里。当微控制器准备好发送数据时,这个分布式 16 位寄存器循环位移 8 个位,这样就有效地在微控制器与外设之间交换了数据。在某些情况下,这种循环位移是不完全的,因为数据可能只是从微控制器到外设或从外设到微控制器。

一个典型的 SPI 系统包括一个主微控制器和一个或多个从属的外设。也有一些系统有两个或多个主微控制器,但这种配置很少见。很多制造商都生产同步串行外设,为了兼容大多数外设,SPI 系统可以选择移位率、主从模式以及时钟的极性、相位等。MC68HC08 系列微控制器总线不可扩展,这就使很多用户感到不便。其实用户可以通过 SPI 系统进行串行扩展,带上 RAM、ROM、A/D、LCD 控制器等很多外设,而且比总线扩展更简单。

SPI 的功能框图如图 8-18 所示。

SPI 特性描述如下:

(1) 全双工,三线同步传送;

(2) 主机或从机工作方式;

(3) 使用相互独立的发送和接收数据寄存器实现双缓存操作;

(4) 4 种可编程主机移位频率(最大可达总线频率的二分之一);

(5) 从机的最大移位频率等于总线频率;

(6) 可编程时钟极性和相位;

(7) 发送结束中断标志 SPTE;

(8) 接收结束中断标志 SPRF;

(9) 工作方式设置错误中断标志 MODF;

(10) 溢出错误中断标志 OVRF;

(11) 可编程线或模式;

(12) 与 I^2C 兼容,一个 SPI 子系统可以在软件控制下构成复杂或简单配置;

(13) 一个主微控制器和几个从微控制器相互连接;

(14) 几个微控制器相互连接构成一个多主机系统;

(15) 一个主微控制器和一个或多个从属外设相互连接。

SPI 系统最常见的应用是一个微控制器做主机。在这种配置下,主机发起并控制数据的传送和流向,只有在主机发出通知后,从属设备才能从主机读取数据或向主机发送数据。本章的所有讨论都建立在这种配置之下。

8.3.1 SPI 的数据流动

在 SPI 的应用中,不需要为每次传送数据特别指出数据流动的方向,因为主机用两根分立的引脚来同时接收和发送数据。分布式移位寄存器操作如图 8-19 所示,在一次 SPI

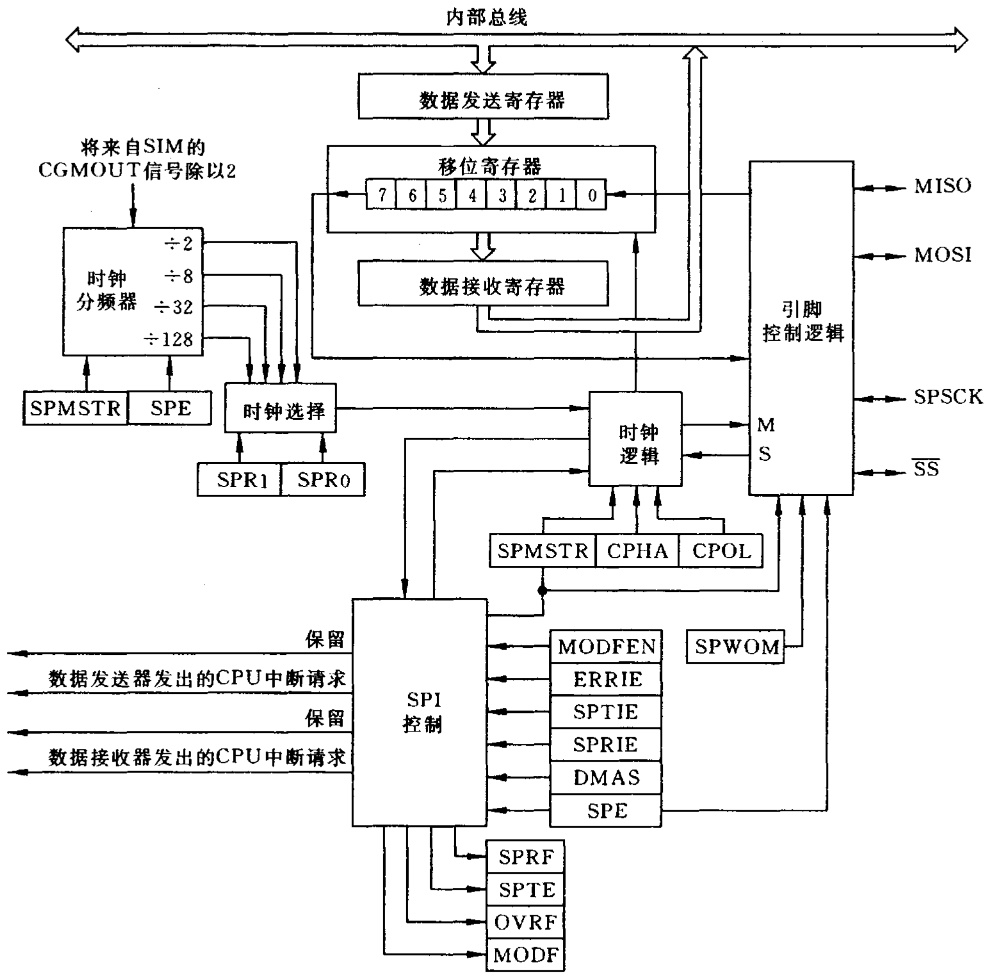


图 8-18　SPI 功能框图

传送发生时，一个 8 位字节通过主出从入 MOSI 引脚移位输出，同时另一个 8 位字节从主入从出 MISO 引脚输入。这个传送过程可以理解为主机的 8 位移位寄存器和从机电路联合组成了一个 16 位循环移位寄存器。在每次传送发生时，这个 16 位分布式移位寄存器循环移动 8 位，这样就完成了主机和外设之间的数据交换。

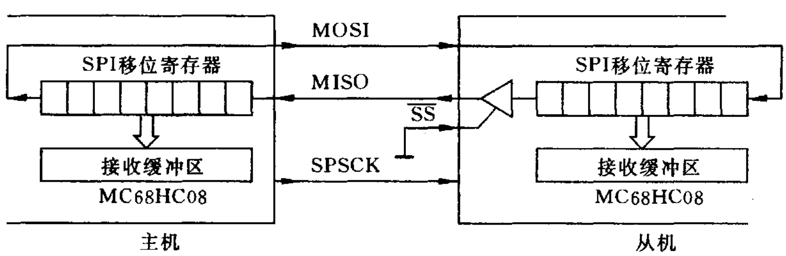


图 8-19　分布式移位寄存器操作图解

有很多简单的设备只能从主机接收数据或只能向主机发送数据，例如串转并移位寄存器就只有一个 8 位的输出端口。假如一个有 SPI 系统的微控制器发起一次传送，向这个移位寄存器发送一个 8 位的数据，由于这个移位寄存器不能向主机回送任何数据，所以主机程序可以简单地忽略掉接收到的数据，然后结束这次传送。

8.3.2 SPI 功能

SPI 系统用 D 端口的 4 根 I/O 引脚来实现数据传送：串行时钟 SPSCK/PD3、主入从出 MISO/PD1、从入主出 MOSI/PD2 和低有效从机片选 $\overline{SS}$/PD0。当不用 SPI 功能时，这 4 根引脚就被配置成通用并行输入线。

在主机模式中，主机启动逻辑从 CPU 接收一个输入（CPU 写到 SPI 数据寄存器的数据）并发出基于内部处理器时钟的 SPSCK，串行时钟也用来控制状态控制器即 8 位移位寄存器。CPU 写数据到 SPI 数据寄存器 SPDR，这个并行数据立刻被装入 8 位移位寄存器并移位输出到 MOSI 引脚，通过从属设备的输入线到从机。同时从机（或外设）数据通过 MISO 引脚输入到微控制器内部的 8 位移位寄存器，第 8 个移位传送后，数据并行传送到 SPI 接收数据寄存器，并等待 CPU 读周期的到来。一个传送结束后，SPRF 或 SPTE 状态标志被主机和外设置位。

在从机模式中，从机的启动逻辑等待$\overline{SS}$引脚被拉低，然后通过 SPSCK 与主机同步。从机通过串行移位寄存器从 MOSI 线接收数据并传送到 SPI 接收数据寄存器。当写周期来到时，从机的 CPU 从内部数据总线将已经写到 SPDR 中的数据装入 8 位移位寄存器，然后等待从主机来的时钟，把数据移位送到 MISO 引脚，随后发送到主机。

8.3.3 SPI 引脚

1. 串行数据引脚

主入从出 MISO 和主出从入 MOSI 用来串行发送和接收数据，数据格式为 MSB 在前，LSB 在后。当 SPI 被置成主机模式时，MISO 是数据输入线，MOSI 是数据输出线。在主机模式下，在 SPI 控制寄存器的第 5 位 SPMSTR 控制位被程序置成 1 之后，允许主机输出数据到 MOSI 引脚。当 SPI 被置成从机模式时，这两根线正好颠倒一下角色，即 MISO 变成从机数据输出线，而 MOSI 变成从机数据输入线。

图 8-20 所示的时序图给出了数据线与 SPSCK 之间的关系。根据控制位 CPHA 和 CPOL 的不同，一共有 4 种可能的时序。从时序图上可以看出，CPOL 为 1 或 0 时，时钟信号 SPSCK 的出现方式正好是反相关系；CPHA 用来选择两种不同的定时协议中的一种，以允许 SPI 系统与各种同步外设通信。

2. 串行时钟

串行时钟 SPSCK 是数据流动的同步信号，通过 MOSI 和 MISO 对设备的数据输入、

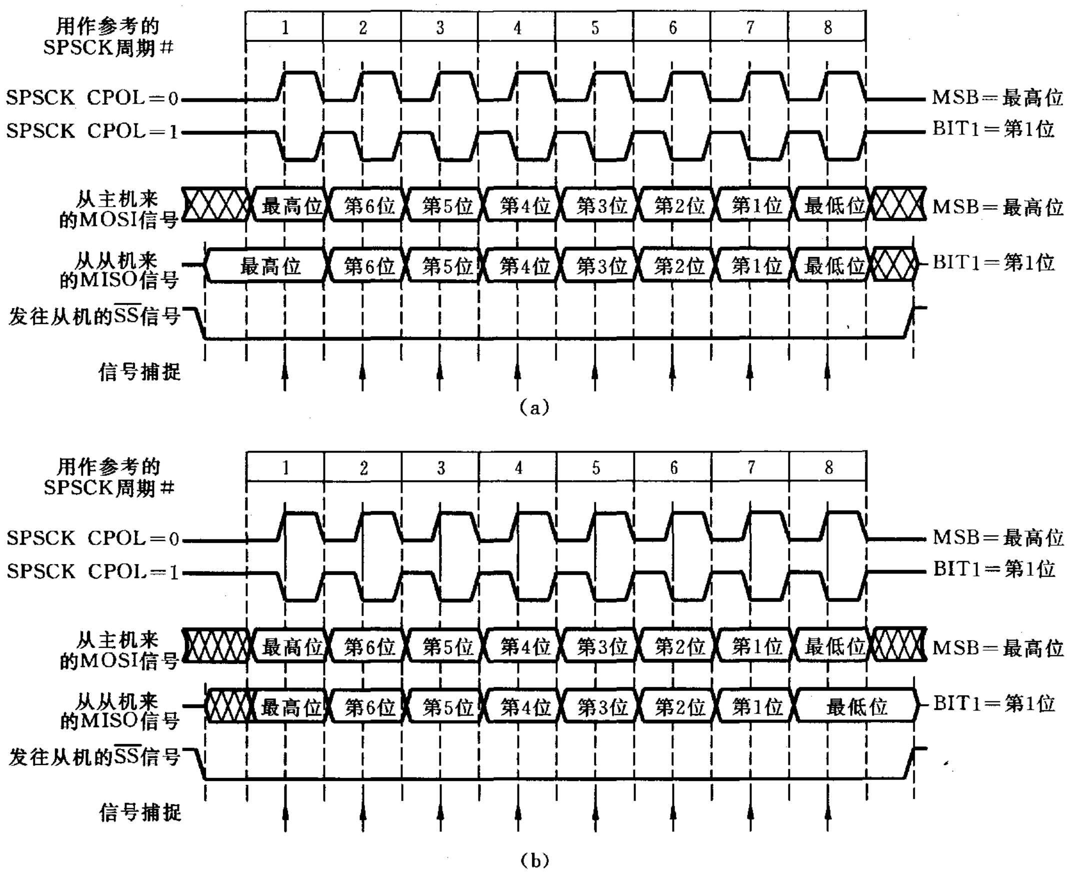


图 8-20 数据/时钟时序图

(a) CPHA=0；(b) CPHA=1

输出进行同步。SPSCK 信号只能由处于主机模式的微控制器发出，对处于从机地位的微控制器或外设，SPSCK 为输入信号。

在主机模式下，SPSCK 信号来源于内部微控制器总线时钟。当主机发起一次传送时，SPSCK 线上会自动产生 8 个时钟周期。在主机与从机之间，数据移位发生在 SPSCK 信号的一个跳变上，待数据稳定后在另外一个跳变进行采样。SPI 状态控制寄存器 SPSCR 的 SPR0 和 SPR1 两位选择 SPSCK 的时钟频率，共有 4 种选择。主机和从机之间必须遵守同样的时序模式，SPI 控制寄存器 SPCR 中的 CPOL 和 CPHA 可以控制时序模式。

3. 外设片选

$\overline{SS}$引脚的使用对于主机和从机是不同的。在从机模式下，$\overline{SS}$是一次数据传送开始前允许从机 SPI 工作的片选信号；对于主机模式，一般都要将$\overline{SS}$线上拉到高电平。

8.3.4 SPI 寄存器

SPI 系统提供了控制、状态和数据 3 个寄存器。这些寄存器包括 SPI 控制寄存器 SPCR(地址 $0010)、SPI 状态控制寄存器 SPSCR(地址 $0011)和 SPI 数据寄存器 SPDR(地址 $0012)。下面逐一介绍。

1. SPI 控制寄存器

在大多数系统中，这个寄存器只能在复位初始化 SPI 系统时写一次，具体各位如图 8-21 所示。

	位	7	6	5	4	3	2	1	0	
$0010		SPRIE	DMAS	SPMSTR	CPOL	CPHA	SPWOM	SPE	SPTIE	SPCR
	复位	0	0	1	0	1	0	0	0	

图 8-21 SPI 控制寄存器

图 8-21 中：

(1) SPRIE 为 SPI 接收中断允许位。

1＝允许 SPI 在 SPRF 位(SPSCR 中的一位)被置成 1 时产生中断请求。

0＝禁止 SPI 在 SPRF 位被置成 1 时产生中断请求(大多数系统采用的配置)。

(2) DMAS 为 DMA 选择位。目前该位在 MC68HC08 系列微控制器的 SPI 模块中并没有意义，在 CPU 做读取操作时其值始终为 0。

0＝禁止 SPRF 和 SPTE 的 DMA 服务请求。

(3) SPMSTR 为 SPI 主机位。

1＝选择 SPI 主机模式。

0＝选择 SPI 从机模式。

(4) CPOL 为时钟奇偶位。

1＝选择低电平有效时钟，SPSCK 空闲状态为高电平。

0＝选择高电平有效时钟，SPSCK 空闲状态为低电平。

(5) CPHA 为时钟相位位。

1＝从机以 SPSCK 的第一次沿跳变为移位开始信号。在多个字节的连续传送过程中，从机的$\overline{SS}$引脚信号可始终保持为低电平。

0＝从机以$\overline{SS}$的下降沿作为移位开始信号。在 SPSCK 的第一次沿跳变启动第一次数据采样，因此在多个字节的连续传送过程中，从机的$\overline{SS}$引脚信号需要不断地恢复为高电平，来为每一个字节的传送产生移位开始信号($\overline{SS}$的下降沿)。

(6) SPWOM 为 SPI 线或模式位。

1＝设置 SPSCK、MOSI 和 MISO 引脚为线或模式。

0＝设置 SPSCK、MOSI 和 MISO 引脚为普通推拉模式。

(7) SPE 为 SPI 允许位。

1＝允许 SPI 模块。

0＝禁止 SPI 模块。

(8) SPTIE 为 SPI 发送中断允许位。

1＝允许 SPI 在 SPTE 位(SPSCR 中的一位)被置成 1 时产生中断请求。

0＝禁止 SPI 在 SPTE 位被置成 1 时产生中断请求(大多数系统采用的配置)。

2. SPI 状态控制寄存器

SPI 状态控制寄存器有 4 个状态标志，用来指示 SPI 系统设置和数据传送过程中的状态。这些状态标志被 SPI 事件自动置位，由软件进行清零，在复位时自动清零。SPI 状态控制寄存器同时还有设置 SPSCK 的时钟频率，允许/禁止错误中断等功能。

具体各位如图 8-22 所示。

	7	6	5	4	3	2	1	0	
位	7	6	5	4	3	2	1	0	
$0011	SPRF	ERRIE	OVRF	MODF	SPTE	MODFEN	SPR1	SPR0	SPSCR
复位	0	0	0	0	1	0	0	0	

图 8-22　SPI 状态控制寄存器

图 8-22 中：

(1) SPRF 为 SPI 接收结束位。当微控制器和外部设备之间的数据传递结束后，数据从移位寄存器进入接收数据寄存器中，微控制器将 SPRF 置成 1 以通知用户接收结束。SPRF 被置成 1 后，用户在程序里读一次 SPSCR 寄存器，再读一次 SPI 数据寄存器后，SPRF 位就被自动清零。

1＝接收数据寄存器满。

0＝接收数据寄存器未满。

(2) ERRIE 为错误中断允许位。

1＝允许 MODF 和 OVRF 产生 CPU 中断请求。

0＝禁止 MODF 和 OVRF 产生 CPU 中断请求。

(3) OVRF 为溢出标志位。当接收数据寄存器中的数据还没有被读取时，如果有新的数据进入移位寄存器，溢出标志位就会被置 1。OVRF 被置成 1 后，用户在程序里读一次 SPSCR 寄存器，再读一次接收数据寄存器后，OVRF 位就被自动清零。

1＝溢出。

0＝未溢出。

(4) MODF 为模式设置错误标志位。当 MODFEN 为 1，且$\overline{SS}$引脚信号与 SPI 当前工作方式(主机或从机)不符时，具体地说，就是当从机的$\overline{SS}$为高电平，或是主机的$\overline{SS}$为低电平时，MODF 会被置 1。MODF 被置成 1 后，用户在程序里读一次 SPSCR 寄存器，再对 SPCR 做写操作，MODF 位就被自动清零。

1＝$\overline{SS}$引脚逻辑电平不正确。

0＝$\overline{SS}$引脚逻辑电平正确。

(5) SPTE 为 SPI 发送结束位。在微控制器和外部设备之间的数据传递过程中，数

据从发送数据寄存器进入移位寄存器中，微控制器将 SPTE 置成 1 以通知用户发送结束。SPTE 被置成 1 后，只要用户在程序里向发送数据寄存器中写入新的数据，SPTE 位就被自动清零。

1＝发送数据寄存器空。

0＝发送数据寄存器未空。

(6) MODFEN 为模式设置错误检测允许位。MODFEN 用于允许或禁止 MODF 标志的置 1。如果 MODF 已经被置起，将 MODFEN 清零并不会使 MODF 也被清零。对于主机而言，如果 MODFEN 为 0，$\overline{SS}$引脚就可以用作普通的输入/输出引脚。

1＝允许 MODF 标志的置 1。

0＝禁止 MODF 标志的置 1。

(7) SPR1/SPR0 为 SPI 移位频率选择位。当微控制器处于主机模式下，这两位控制串行时钟 SPSCK 的频率，具体请见表 8-6。在从机模式下，这两位没有意义。

计算公式为

SPI 时钟频率＝(CGM 基准时钟输出)/(2×时钟分频数)

表 8-6　SPI 时钟频率选择

SPR 位		内部时钟分频数	CGM 基准时钟输出 MHz	
SPR1	SPR0		4.0	2.0
0	0	2	1.000MHz	500.0kHz
0	1	8	250.0kHz	125.0kHz
1	0	32	62.5kHz	31.25kHz
1	1	128	15.625kHz	7.8125kHz

3. SPI 数据寄存器

SPI 数据寄存器如图 8-23 所示，它由两个独立的发送数据寄存器(只写)和接收数据寄存器(只读)组成，二者共用一个内存地址($0012)。当 CPU 向 SPDR 执行写操作时，数据被写入发送数据寄存器中；当 CPU 向 SPDR 执行读操作时，接收数据寄存器中的数据被读出。采用两个独立的数据寄存器使得它们可以同时储存不同的数据，也就是说向 SPDR 写入的数据并不能从 SPDR 中读出。

	位	7	6	5	4	3	2	1	0	
$0012	读	R7	R6	R5	R4	R3	R2	R1	R0	SPDR
	写	T7	T6	T5	T4	T3	T2	T1	T0	

图 8-23　SPI 数据寄存器

在实际的工作过程中，数据一旦写入发送数据寄存器，就会被立即送到移位寄存器并将 SPTE 置 1，这时虽然数据仍需要 8 个 SPSCK 周期才能发送完成，用户却已经可以向发送数据寄存器写入新的数据了。而只要第一个数据发送完成，新的数据会被立刻从发

送数据寄存器中读到移位寄存器中并开始第二次数据发送，同时 SPTE 将再次置 1，这时用户又可以写入第三个数据了。这样的好处是将相邻字节发送的间隔缩短到几乎为 0，大大提高了 SPI 的发送效率。

8.3.5 SPI 应用举例

本书介绍 MC68HC908GP32 在线编程开发系统中的一个液晶显示模块的实验，其中使用的接口模块就是 SPI。下面以这个实验为例子，简单介绍如何用 SPI 实现数据的传递。

该实验的相关硬件连接见图 8-24。

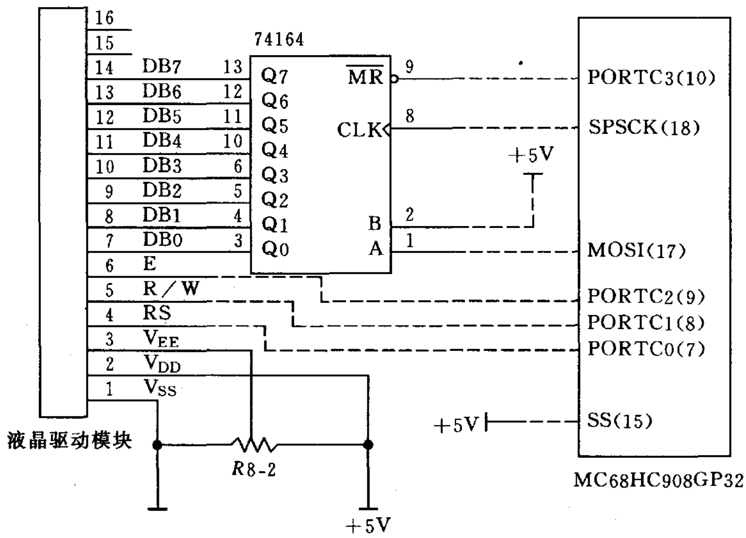


图 8-24 SPI 应用举例——硬件连接

实验中使用的是单排字符液晶，其控制模块有 16 支引脚，但在 MC68HC908GP32 在线编程开发系统中仅有 14 支引脚有信号定义，分别是地线（V_{SS}）、电源线（V_{DD}）、驱动电压线（V_{EE}）、寄存器选择信号（$\overline{RS}$）、读/写信号（R/W）、片选信号（E）和 8 位数据线（DB0～DB7）。在实验中，我们用 74164 将液晶的 8 位并行数据输入方式转化为 SPI 串行输入方式。

程序中对 SPI 的初始化代码如下：

```
    LDA     #%00111010      ;禁止中断，主机方式，时钟低电平有效
SPSCK
                            ;启动移位脉冲，普通推拉模式，允许 SPI
    STA     SPCR
    LDA     #%00000000      ;禁止中断，禁止错误检测，时钟分频数为 2
    STA     SPSCR
```

使用 SPI 发送数据的代码如下：

```
BRCLR    B_SPTE,SPSCR,*          ;确认发送数据寄存器为空
LDA      #%00111000              ;向液晶送出命令字"000011****"
STA      SPDR
```

由于篇幅有限，下面只提供部分源程序清单，整个实验的源程序请参看 MC68HC908GP32IDK 的相关资料和文档。

```
***==========MC68HC908GP32 寄存器定义
PORTC         EQU      $0002
B_RS          EQU      0
B_RW          EQU      1
B_E           EQU      2
B_MR          EQU      3
DDRC          EQU      $0006 *
*==========SPI=============
*
SPCR          EQU      $0010
SPSCR         EQU      $0011
B_SPTE        EQU      3
SPDR          EQU      $0012
*
              ORG      $00B0
*
OUT_LCD_BUFFER  RMB    1
LCD_MASK      RMB      4
LCD_ADD       RMB      1
BUFFER        RMB      1
LCD_BUFFER    RMB      16
*
              ORG      $8000
*
INIT:
              LDA      #%00111010      ;SPI 初始化
              STA      SPCR
              LDA      #%00000000
              STA      SPSCR
*
              LDA      DDRC            ;初始化 E、R/W 和RS控制信号
              ORA      #%00001111
              STA      DDRC
*
```

```
            JSR     CLR_RD              ;74164 归零
*
            JSR     INIT_LCD            ;LCD 初始化
*
            LDHX    # $00               ;设置 LCD 显示内容
L_L_SETBUF:
            LDA     CORP,X
            STA     LCD_BUFFER,X
            AIX     #1
            CPHX    # $10
            BNE     L_L_SETBUF
*
            JSR     SHOW_LCD_BUFFER     ;将数据送到 LCD
*
INIT_NOLCD:
            BRA *                       ;主程序结束
*
CORP        FCB " TSINGHUA MAC "        ;LCD 显示的字符
*=========================================
*
*           INIT_LCD
*
*=========================================
*
INIT_LCD:
            LDHX    # $44E              ;等待 4.5ms
INILCD1_4_MS:
            AIX     #-1
            CPHX    #0
            BNE     INILCD1_4_MS
*
            BCLR    B_RS,PORTC          ;RS=0
            BCLR    B_RW,PORTC          ;RW=0,向 LCD 写入数据
*
            BRCLR   B_SPTE,SPSCR,*      ;确认发送数据寄存器为空
            JSR     CLR_RD              ;74164 归零
            LDA     #%00111000          ;向液晶送出命令字"000011****"
            STA     SPDR
            BRCLR   B_SPTE,SPSCR,*
            NOP
            NOP
```

```
        BSET        B_E,PORTC        ;让 E 出现一个正脉冲
        NOP
        NOP
        BCLR        B_E,PORTC
*
……
```

8.4 A/D 转换器

部分 MC68HC08 系列的微控制器具有 A/D 模块，例如 MC68HC908GP32 就具有 8 路 8 位 A/D 转换器，并与并行 I/O 口 PORTB 复用 8 个芯片引脚。

使用片内 A/D 转换器需要注意的问题主要有两个：一是选择适当的 A/D 通道，二是正确地设置 A/D 的转换频率。A/D 转换器共有 16 路输入通道，其中有 10 路有效通道，5 路无效通道，还有 1 路用于关闭 A/D 电源。在 10 路有效输入通道中，有 8 路是复用普通的 I/O 口输入转换电压，还有 2 路是利用标准的内部电压对 A/D 转换器的功能进行校验和测试。A/D 转换器的频率必须在 1MHz 左右，用户需要选择适当的时钟源和分频数以满足这一要求。

对 A/D 转换器的操作是通过 A/D 状态和控制寄存器(ADSCR)、A/D 结果寄存器(ADR)和 A/D 时钟寄存器(ADCLK)来完成的，下面分别对各个寄存器做一下介绍。

8.4.1 A/D 状态与控制寄存器

A/D 状态与控制寄存器如图 8-25 所示。

	位	7	6	5	4	3	2	1	0	
$003C		COCO/IDMAS	AIEN	ADCO	ADCH4	ADCH3	ADCH2	ADCH1	ADCH0	ADSCR
	复位	0	0	0	1	1	1	1	1	

图 8-25　A/D 状态控制寄存器

图 8-25 中：

(1) AIEN 为 A/D 中断允许位。

1＝允许 A/D 中断。

0＝禁止 A/D 中断。

(2) COCO/IDMAS 为 A/D 完成标志位，或是 DMA 中断选择位。这一位的功能由 AIEN 的值决定。当 AIEN＝0 时，A/D 中断被禁止，用户只能通过查询标志来确认 A/D 转换是否结束，此时 COCO/IDMAS 便作为 A/D 完成标志来使用，它的置位就标志 A/D 转换结束；当 AIEN＝1 时，A/D 中断被允许，一旦 A/D 转换完成，相应的 A/D 中断请求会自动发往 CPU 或 DMA，此时 COCO/IDMAS 则用于选择使用 CPU 中断还是 DMA 中断来处理 A/D 中断请求。DMA 是 MC68HC08 系列微控制器的一个专门模块，但在

MC68HC908GP32、MC68HC908JL3 等常用微控制器中并没有这一模块，所以 DMA 中断请求的用法不在本书中详述。

当 AIEN＝0 时，1＝A/D 转换结束；0＝A/D 转换未结束。

当 AIEN＝1 时，1＝选择 DMA 中断；0＝选择 CPU 中断。

(3) ADCO 为 A/D 连续转换控制位。

1＝连续转换，A/D 转换器会不断地将新的转换值写入 A/D 数据寄存器中。

0＝单次转换。

(4) ADCH4～ADCH0 为 AD 通道选择位。这 5 位用于选择 A/D 的输入通道，其具体意义请见表 8-7。

表 8-7　A/D 输入通道选择

ADCH4	ADCH3	ADCH2	ADCH1	ADCH0	Input Select
0	0	0	0	0	PTB0/AD0
0	0	0	0	1	PTB1/AD1
0	0	0	1	0	PTB2/AD2
0	0	0	1	1	PTB3/AD3
0	0	1	0	0	PTB4/AD4
0	0	1	0	1	PTB5/AD5
0	0	1	1	0	PTB6/AD6
0	0	1	1	1	PTB7/AD7
↓	↓	↓	↓	↓	Reserved
1	1	0	1	1	Reserved
1	1	1	0	0	Reserved
1	1	1	0	1	V_{REFH}
1	1	1	1	0	V_{SSAD}
1	1	1	1	1	ADC power off

8.4.2 A/D 数据寄存器

A/D 数据寄存器如图 8-26 所示，它是只读存储器，它的值由 A/D 转换器在每一次转换完成后自动更新。

	位	7	6	5	4	3	2	1	0	
$003D		AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	ADR
	复位	0	0	0	0	0	0	0	0	

图 8-26　A/D 数据寄存器

8.4.3 A/D 时钟寄存器

A/D 时钟寄存器用于选择 A/D 转换器的时钟频率。

A/D 时钟频率可用以下公式计算：

A/D 时钟频率＝A/D 源时钟频率/分频数

为了得到可靠的 A/D 转换结果，需要为 A/D 转换器选择合适的时钟源和分频数，使 A/D 时钟频率在 1MHz 左右。

A/D 时钟寄存器如图 8-27 所示。图中：

	位	7	6	5	4	3	2	1	0	
$003E		ADIV2	ADIV1	ADIV0	ADICLK	0	0	0	0	ADCLK
	复位	0	0	0	0	0	0	0	0	

图 8-27 A/D 时钟寄存器

(1) ADIV2～ADIV0 为 A/D 时钟分频选择。A/D 时钟分频数如表 8-8 所示。表中 X＝任意值。

表 8-8 A/D 时钟分频数

ADIV2	ADIV1	ADIV0	分频数
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	X	X	16

(2) ADICLK 为 A/D 时钟源选择。

1＝选择内部总线时钟。

0＝选择外部晶振时钟。

例如当选择 8MHz 的内部总线时钟为 A/D 时钟源时，应同时设置时钟分频数为 8，这样 ADICLK 为 1，ADIV 为 3，ADCLK 寄存器的值应该是 $70。

第 9 章　用 C 语言开发 MC68HC08 微控制器系统应用程序

9.1　C 语言是开发微控制器应用系统软件的有力工具

C 语言是一种通用的程序设计语言，它有丰富的运算符和表达式、先进的控制流和数据结构，是软件开发的有力工具。C 语言是 20 多年来，随 Unix 操作系统的研制而发展起来的。Unix 也反映了 C 语言的成功，在 13000 行源码中，汇编语言只有 800 行。C 语言算不上很高级的语言，它与计算机处理的是同一类型的对象，即字符、数和地址，它比其他高级语言显得更像汇编语言。

尽量使用高级语言写软件才能达到事半功倍的效果。

MC68HC08 是由 MC68HC05 进化而来的，与 MC68HC05 向上兼容。从 MC68HC05 到 MC68HC08，性能方面最重要的改进就是把 MC68HC05 的固定式堆栈指针变成 MC68HC08 的浮动式堆栈指针。正因为有了这一重大改进，高级语言才有可能用得上。因为在 C 语言中，参数的传递、函数中的局部变量等都要从栈空间中得到。像 MC68HC05 那样栈是死的，且只有 64B，不可能使用高级语言。而到了 MC68HC08，堆栈指针活了，高级语言自然上得去了。如果还是使用汇编语言来开发应用程序，MC68HC08 的这一大优势就得不到发挥。

9.1.1　使用 C 语言编写应用程序

C 语言是与硬件无关的通用程序设计语言。C 语言本身并不包括输入输出语句，但 C 语言需要通过输入输出语句同硬件打交道，而只有输入输出语句中最底层的几个函数才与硬件设计有关。换言之，只要程序设计者提供给 C 编译器几个最基本的 I/O 函数，在程序设计中使用 C 语言就没有什么问题了。

用 C 语言写应用程序至少有以下 5 方面的好处。

(1) 系统可以在其他计算机上仿真。由于微控制器的使用，硬件设计变得越来越容易，相比之下，软件开发的工作量往往会比硬件设计大得多。在其他计算机上，例如在 IBM-PC 上用 C 语言编写 MC68HC08 微控制器系统的应用程序，并在 PC 机上仿真整个系统，可以使 MC68HC08 微控制器系统的软件开发与硬件设计同时进行，甚至在硬件设计之前就开始软件的开发工作。

(2) 应用程序的可移植性。考虑到将来产品更新换代，需要更换比 MC68HC08 更强有力的微控制器时，用 C 语言写的程序往往可以直接移植，只需要重新编译一下就可以了，而如果整个应用程序都用汇编语言来编写，将来的移植几乎是不可能的。

(3) 便于程序调试。应用程序的调试可以直接使用 C 语言，例如 C 语言中一个典型

的按格式输出函数 printf(),是程序调试的有力工具,程序中的各个参数,包括一些中间变量都可以用这个函数在屏幕上显示出来,这比汇编语言程序调试中设断点的方法轻松多了。

(4) C 语言库函数丰富。C 语言提供许多有用的库函数,例如数学运算、码的转换、各种格式的输入输出等。为应用编写的一些汇编语言程序也可以放在库里,作为开发下一个产品的资源。

(5) 源程序易读,易于修改。

9.1.2 使用汇编语言编写与硬件相关的程序

哪部分程序适宜用汇编语言编写呢?

与系统硬件相关的部分,虽然也可以写成 C 语言的形式,但我们认为,还是使用汇编语言来写更为直观、更方便。就 MC68HC08 而言,用汇编语言写起来比较容易的有以下 3 个内容。

(1) 系统的初始化,包括所有应初始化的寄存器,各口、各位在系统中的定义,栈指针的设置等,以建立 C 语言程序运行的环境。

(2) 中断向量的初始化,中断服务程序的入口参数和出口参数,开中断关中断等,而中断服务本身则可用 C 语言编写,用汇编程序调用 C 语言写的子程序完成中断服务。

(3) 用汇编语言写输入口、输出口的输入输出函数,在 C 语言程序中调用这些函数。

总之,用汇编语言写与硬件有关部分的程序,用 C 语言写与硬件无关部分的程序,处理好两部分程序之间的参数传递是成功的关键。

对于 MC68HC08 微控制器系统,与硬件相关部分的程序不会很大,一般不会超过 2kB 机器码。如果整个程序大于 4kB,我们主张使用 C 语言编写应用程序。应用程序越大,使用高级语言的好处就越显著,不必担心 C 语言的效率或速度。对于熟悉汇编语言的编程员,C 语言就是与机器无关的汇编语言,写过一些程序以后就会感觉到,一面写 C 语言程序,汇编程序的大致模样就浮现在眼前。

9.2 交叉编译和 C 语言程序运行环境的建立

所谓交叉编译,是指在一台大一些、用起来更方便的计算机上,编译产生目标系统的机器码。目标系统所用的 CPU 及指令系统与所用的计算机的 CPU 类型完全不同。例如,用 IBM-PC 机的交叉 C 语言编译器产生 MC68HC08 目标码的过程就是交叉编译过程。

C 语言本身不但与硬件无关,而且与操作系统无关,C 语言的交叉编译器在遇到任何 I/O 时,便留给操作系统。MC68HC08 在没有操作系统时,便将 I/O 任务留给与硬件相关的函数。分析 C 语言是如何工作的,有利于理解 C 语言。

我们知道,C 语言的主程序是从 main()开始的,也就是说,main()这个函数可以看成是应用程序本身,要使 main()能运行,就要建立一个程序运行环境。什么是 C 语言程序

的运行环境呢？可以通过分析一个空程序来了解C语言的运行环境。

请读者注意这里使用的分析方法，这个方法对建立其他微控制器的开发环境也是适用的。

```
Function1()
{
}
```

这是一个空函数，什么也不做，我们只是定义了一个名叫 Function1 的函数。编译这个函数，我们得到了长度为一个字节的代码。编译过程中或许会出现一个警告性错误：函数无返回值。忽略这个警告性错误，这个函数只有一条指令：

```
RTS 子程序返回
```

显然，引用这个子程序时，一定是用子程序调用指令：

```
JSR     function1
```

让我们来分析 main()这个函数，从而得到如何建立一个应用程序的运行环境：

```
main()
{
}
```

仍然是什么也不做。编译这个程序，并链接，我们发现，最终生成的机器码比一个字节多。main() 作为一个函数，仍然只生成一个字节的返回指令，其他代码则是建立程序运行环境用的。

对于通过操作系统和I/O打交道的C语言程序，main()是作为一个任务递交给操作系统去完成其运行的。运行环境的建立相对要复杂一些。首先要对 main(argc ，argv)中的命令行参数进行解释，C语言允许在括号内代入两个命令行参数：参数个数和参数字符串数组指针。

其次，运行C语言程序，允许在I/O中使用输入输出再定向，这是操作系统的事。

对于用微控制器构成的系统，以上两种功能都可以略去，而在多数情况下，微控制器构成的控制系统，并不需要文件系统，这就使得运行环境的建立大大简化。

实际上，main()程序的运行环境可以简化到顺序完成以下3个任务。

(1) 设置栈指针初值；

(2) 调用函数 main()使之执行；

(3) 给出 main()完成后的出口，即执行 exit()。

因为只要设定了有效的栈指针和栈空间，函数调用语句JSR才有可能实现，此时调用点的PC值才能在栈中保留。

main()可以看成是整个应用程序，它可以由一系列子程序调用完成，第一个应该调用的子程序，显然是系统的初始化子程序，这个系统初始化子程序可以是许多子程序模块中的一个，这部分用汇编语言或用C语言写都可以。

exit()是C语言的一个标准函数，旨在告诉系统，main()这个程序运行结束后的出口

在哪里，实际的应用程序也许根本用不着有出口，但必须在建立C语言程序运行环境时给出完整的exit()函数。exit()函数至少在程序调试过程中是非常有用的，可以把监控程序的热启动地址作为程序main()的出口。

堆栈指针的设置是与硬件相关的，往往设在RAM区的高端，用栈顶的地址STACK代表栈顶地址。建立main()程序运行环境的过程如下：

```
LDHX    #STACK
TXS
JSR     main()
JSR     exit()
```

9.3 应用程序的构成与模块化程序结构

在建立C语言程序运行环境时，可以把main()看成整个应用程序，而实际上应用程序是由许多个子程序或许多个函数构成的。把应用程序构造成多个小的模块，每个模块由若干函数或若干子程序组成。这些模块一个一个分别调试，最后连接在一起是写应用程序的基本方法。每个模块可以用C语言写，也可以用汇编语言写，哪个来得方便就用哪个写，每个模块不超过一两页纸，每个模块都是一个单独的文件。这样调试、修改都很方便，余下来的问题便是建立运行环境问题了。

每个函数的运行环境与上面讲到main()函数的运行环境实际上是一样的，如忽略参数的传递(注意，这里参数的传递是不能忽略的，参数传递办法在9.5节中专门讲)，就只剩下一个栈指针的问题了。这个问题实际上是要告诉运行环境，本函数有多大的栈空间可以使用。这牵扯到该函数有多大的局部变量区，有多少层子程序嵌套，如果我们一开始就把栈区开得足够大，每个子程序的运行一般就不会有什么问题。

估算栈空间的大小是很容易的，因为定义了多少个局部变量，写程序时是清楚的，函数调用的层次也能数得出来。对于像printf()这样的函数，经过多少层才调用到底层的函数putchar()，用于输出一个字符，用户难以准确判断。假定它有10层调用，也不过占去20个字节的栈空间。另外要考虑到的是：一次中断过程用于保护现场的栈空间是5个字节。这在MC68HC08中是自动进行的。5再乘以中断嵌套层数，就是用于中断的栈空间。除非程序中有所谓递归调用，即函数自身调用自身，在微控制器系统中栈空间所占内存是相当有限的。

总之，将应用程序写成若干个模块，每个模块中可以有若干个函数或子程序。这些模块如是用C语言写的则称为函数，用汇编语言写的叫做子程序，这样听起来更习惯一些。问题的实质是，如果用汇编语言写的子程序和用C语言写的函数有完全相同的结构和参数传递办法，那么模块本身是用什么语言写的也就无关紧要了。

9.4 全程变量与局部变量

全程变量是为整个程序定义的，在程序运行中自始至终都有效。用全程变量传递参

数，是参数传递的常用方法，在C语言或汇编程序模块中都可以直接使用。函数在没有局部变量、没有参数传递的情况下具有最简单的结构，全程变量作为外部变量可以直接引用。在有局部变量和参数传递的情况下，局部变量是在栈空间定义的，局部变量实际上是子程序要用到的一块内存空间。参数的传递是将值拷贝给函数的形式参数。函数的形式参数是指定义函数时，括号里的那一个或几个变量，它们也是在栈空间里定义的。

9.5 函数的结构与函数间参数的传递

为彻底弄清楚函数的结构和参数传递办法，这里提供一种分析方法，使我们可以用汇编语言写符合C函数格式的子程序。建议读者在使用C语言写应用程序前自己做一遍。

9.2节提到，写一个空函数，C语言编译以后只生成一个子程序返回码RTS。在这个函数的基础上，逐渐地增加内容，每加一句，看它在汇编中形成什么样的码，最后总结出一个完整的函数在汇编中是什么样子。以后的汇编程序就照这个格式写，用C语言写MC68HC08的应用程序的关键问题就解决了。

首先，为了去掉编译过程中出现的警告信息：缺少返回值。我们加上一句return(0)。然后，定义*a*为一全程变量，看函数如何引用全程变量。再把*a*定义成局部变量，看它们之间有什么区别。

C语言程序：

```
/* Global.c */
extern int a;
Global()
{
    a=5;
    return(0);
}
```

编译该函数生成汇编码：

```
        EXTERN  a
        PUBLIC  Global
Global: LDA     #0x05
        STA     a:0x1
        CLRA
        STA     a
        CLRX
        RTS
```

当*a*定义为局部变量时：

```
/* Local.c */
Local()
{
    int a;
    a=5;
    return(0);
}
MEM
```

```
Local:  AIS     #-2     ;MEM for Int a
        LDA     #0x05
        STA     2,SP    ;Assign a
        CLRX
        CLRA
        AIS     #2      ;Release
        RTS
```

注意，这里的AIS #-2和AIS #2的意思是为局部变量*a*在堆栈中分配和释放内

存。如果读者仍然对此感到困惑，可以在C语言中加上

```
int   a,b;   或   int   a,b,c;
```

重复以上编译，对于局部变量的定义、赋值与释放就清楚了。

该例子只说明局部变量的结构，本身没有什么意义，局部变量只在函数内部有意义。它占用了一块栈空间，在子程序返回前又释放掉了。局部变量实质上是子程序要用到的缓冲区，不牵扯参数传递。

下面以对 EEPROM 的写入函数来说明数值参数的传递过程。

我们希望该函数在C语言中有如下形式：

```
prgeeprom (Data,P)
unsigned int Data          /* 要写入的值 */
unsigned int P             /* 写入地址的指针 */
```

在C语言中调用这个函数时，只要把 Data 和 P 的值代入该函数，就可以向 P 指定的 EEPROM 单元写入 Data。

该函数用汇编语言可写成：

```
PRGEEPROM:  PSHA                       ;Save P
            PSHX
            LDA      6,SP              ;Get Data
            STA      Data:0x1
            LDA      5,SP
            STA      Data
            LDA      2,SP              ;Get P
            STA      P:0x1
            STX      P
            JSR      EEPROM            ;subroutine to write
            AIS      #2
            RTS
```

若 Data＝$59，P＝$B605，当在C语言程序中调用函数 PRGEEPROM(0xB605，0x59)时，生成的汇编程序如下：

```
LDHX     #0xB605       ;Store Data in stack
PSHX
PSHH
LDA      #0x59         ;Store P in A and X
CLRX
BSR      PRGEEPROM
```

模块间参数的传递规则可归纳如下：

(1) 对于函数带出的参数，相当于 return(n)中的 n 值。如果 n 是 1 个字节的二进制数(Byte)，会存在 A 寄存器中；如果 n 是 2 个字节的二进制数(Word)，则低位字节存在 A

寄存器中，而高位字节存在 X 寄存器中，否则通过堆栈传递返回值。

(2) 对于带入的参数，如果只有一个，则放在 A 寄存器或 A 和 X 寄存器中。

(3) 如果函数有两个以上的形式参数，调用时，将多于一个的参数由左向右逐个压入栈中，只留最后一个参数在 A 寄存器或 A 和 X 寄存器中。用的编译器不同，最后一个参数的保留方式会有所不同。该函数在取这些参数时，到栈区以 SP 为指针，以偏移量+2，+4…存取。

在本例中，由于函数在入口处将第二个参数 P 压入堆栈，所以此时的(SP+1,SP+2)储存的是第二个参数，(SP+3,SP+4)是调用过程本身用到的子程序返回地址，而(SP+5,SP+6)则是第一个参数。

读者如果对以上参数传递过程还不够清楚的话，不妨用 C 语言写一个简单的函数，编译时看生成的汇编码是什么，读懂这段汇编码，写汇编时照该格式写，写成的汇编模块就可以在 C 语言程序中自由调用。

9.6 程序模块的框架与组织

前面讲到，一段段函数或子程序要组织成模块，每个模块一般只有一两页纸，每个模块为一个文件。下面说明一个应用程序应该有哪些文件，假定应用程序名叫 EXAMPLE，需要建立以下文件：

(1) EXAMPLE. H，用以定义所有的全程变量。

在该文件中用以下格式定义全部全程变量：

```
EXTERN     char      a;
EXTERN     int       b;
EXTERN     char      *p;
EXTERN     double    d;
     ⋮
```

(2) 用 C 语言写的函数模块文件。

例如名 Funcl. c，Func2. c…在这类文件中开头写上

```
#define     EXTERN    extern
#include    "EXAMPLE.H"
```

(3) 用 C 语言写的主程序 EXAMPLE. C。在该文件中定义 EXTERN 为空。

```
#define     EXTERN
#include    "EXAMPLE.H"
main( )
{
⋮
}
```

这样编译时，所有的全程变量将在这里插进来得到认可。

(4) 建立运行环境的汇编文件 castartup. asm。

```
         EXTERN   main        用C语言写的主程序
         EXTERN   INITIAL     用汇编语言写的主程序
         EXTERN   EXIT
Startup: LDHX     # STACK
         TXS
         JSR      INITIAL
         JSR      MAIN
         JMP      EXIT
```

(5) 全部模块 Link 文件 EXAMPLE. LNK。

将所有模块文件名、C 语言库文件逐行列在此文件中,并定义程序起始地址、数据区起始地址和栈顶地址。

9.7 宏调用

前面几节只用了函数调用或子程序调用的概念。函数的调用和子程序调用是同一概念,是指调用前将当前程序计数器 PC 值入栈。函数若成了程序的最后一条指令,那么一定是子程序返回指令 RTS。RTS 指令使栈中保存的 PC 值弹出,程序执行完函数调用后继续向下执行。这样做的好处是子程序得到了复用,程序代码较短。宏调用是指在调用子程序时,不使用 JSR 和 RTS 这一对调用子程序和子程序返回指令,而直接将子程序代码插入到程序中。这样做的优点是执行速度加快了,因为省去了两条指令,缺点是该段程序得不到复用,多次插入相同的代码,程序会变得比较长。而 MC68HC08 微控制器中,ROM 或 Flash 的空间相对较大,当子程序代码较短,对执行时间要求较为苛刻时,不妨使用宏调用。特别是在一些 I/O 操作上,使用宏调用往往比子程序调用方便。

宏调用的实现在 C 语言中是使用#define 语句实现的。多数用于微处理器的 C 语言交叉编译器允许在 C 语言程序中直接插入汇编代码。

例如:

```
#define  HALT               {_asm CLRA; _asm SWI ! {A}, {A+HX+SR};}
#define  HALTX(x)           {_asm LDA x; _asm SWI ! {A}, {A+HX+SR};}
#define  HALT_AND_QUIT      HALTX(#32)
#define  EnableInterrupts   {_asm CLI;}
#define  DisableInterrupts  {_asm SEI;}
```

第 10 章　低功耗模式

MC68HC08 中提供了两种低功耗模式供用户选择使用，这两种方式分别对应微控制器的指令 WAIT 和 STOP，相应的低功耗模式被称为 WAIT 模式和 STOP 模式。

MC68HC08 系列微控制器为静态 CMOS 工艺。CMOS 电路的特点是在静态时功耗极小，功耗主要来自电平变化过程中的瞬态电流。所以时钟频率越高，系统功耗越大。这一点从 10.1.2 节中的功耗—频率曲线图中可以清楚地看出来（以 MC68HC08GP32 为例）。降低功耗的方法是在不需要微控制器工作的时候，例如等待某个信号的输入或中断请求时，关闭系统时钟，使功耗达到最小，进入“睡眠”状态。而当有外部输入需要处理的时候，再让微控制器脱离“睡眠”被“唤醒”，恢复时钟，继续运行指令。如果外部事件不频繁，采用这种方法就可以大大降低系统功耗。例如，如果系统只有 1/2 时间处理外部事件，其余时间保持低功耗状态，则平均功耗就可减低为原来的 1/2。WAIT 和 STOP 指令都是通过关闭内部时钟或外部时钟的方式减小微控制器的功耗。

10.1　WAIT 和 STOP 指令

10.1.1　STOP 指令

在程序中运行 STOP 指令使微控制器处于最小的功耗模式。在 STOP 状态下，关闭微控制器内部时钟，包括 CPU 的时钟和内部总线上的时钟，微控制器内部的一切操作停止。当系统进入 STOP 模式，状态寄存器的中断标志位自动清零时，任何的外中断（IRQ，KBI）及 SCI 等中断都可以“唤醒”微控制器，脱离 STOP 状态。执行 STOP 指令后微控制器中所有的寄存器和 RAM 中的数据保持不变，I/O 管脚的电平也保持不变。

微控制器进入 STOP 模式后，只有中断或外部复位可以使微控制器脱离 STOP 状态。脱离 STOP 状态后，CPU 时钟和总线时钟继续工作，频率等于执行 STOP 指令前的频率。

执行 STOP 指令后的流程如图 10-1 所示。

10.1.2　WAIT 指令

WAIT 指令使微控制器进入低功耗模式。WAIT 模式的功耗比 STOP 模式大。在 WAIT 模式下，内部 CPU 的时钟被关闭，但内部总线时钟并不停止，定时器仍然在工作，并允许定时器中断。所以 WAIT 模式下，微控制器还可以执行和定时器相关的工作。如果关闭 LVI（低电压禁止）模块，功耗还可以进一步降低。与 STOP 方式相同，WAIT 方式下只有中断和外部复位可以唤醒微控制器，此时系统所有的寄存器和 RAM 中的内容保持不变，I/O 管脚的电平也保持不变。

执行 WAIT 指令后的流程如图 10-2 所示。

WAIT 模式下 MC68HC08GP32 的功耗如图 10-3 所示。

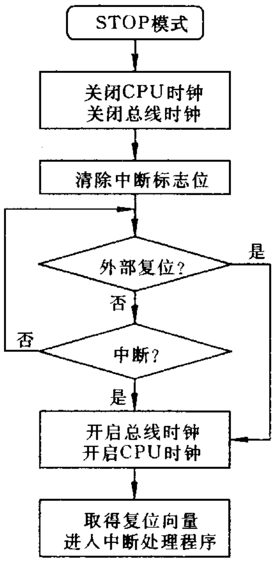


图 10-1 执行 STOP 指令后的流程

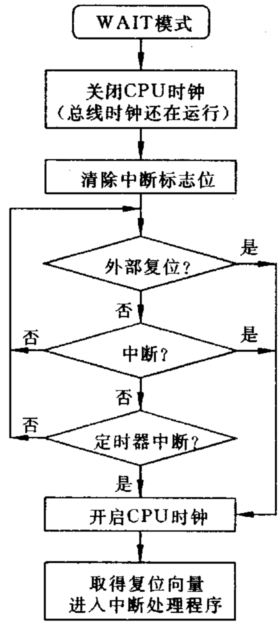


图 10-2 执行 WAIT 指令后的流程

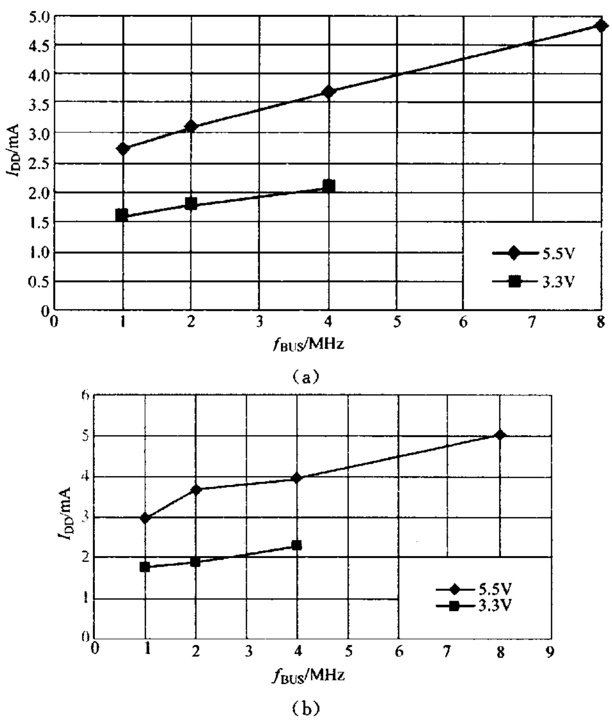


图 10-3 WAIT 模式下 MC68HC08GP32 的功耗

(a) 测试条件:温度为−40~85℃,LVI 模块关闭;

(b) 测试条件:温度为−40~85℃,LVI 模块打开

10.2 唤醒微控制器的事件

当微控制器执行 WAIT 或 STOP 指令后，只有外部事件可以使微控制器退出 WAIT 或 STOP 模式。这些事件包括

(1) 外部复位。当微控制器的复位引脚上出现逻辑 0 将使微控制器复位，并从地址 \$FFFE～\$FFFF 处取得复位向量时，开始执行下一条指令。

(2) 外部中断。微控制器外中断引脚上加一由高到低的电平，微控制器将从地址 \$FFFA～\$FFFB 处取得中断向量并执行中断服务程序。

(3) 看门狗复位。看门狗超时复位将使微控制器复位并从地址 \$FFFE～\$FFFF 处取得复位向量。

(4) 低电压禁止复位。微控制器供电电压低于允许值将使微控制器复位并从地址 \$FFFE～\$FFFF 处取得复位向量。

(5) 时钟发生模块中断。时钟发生模块的锁相环电路的中断请求将从 \$FFF8～\$FFF9 处取得中断向量并执行中断服务程序。

(6) 键盘中断。键盘中断将唤醒微控制器，并从 \$FFE0～\$FFE1 处取得中断向量执行中断程序。

(7) 定时器产生的中断(仅适用于 WAIT 模式，STOP 模式将关闭定时器)。

与定时器 TIM1 有关的中断和向量地址为

\$FFF2～\$FFF3	TIM1 计数器溢出中断；
\$FFF4～\$FFF5	TIM1 通道 1 输入捕捉、输出比较中断；
\$FFF6～\$FFF7	TIM1 通道 0 输入捕捉、输出比较中断。

与定时器 TIM2 有关的中断和向量地址为

\$FFEC～\$FFED	TIM2 计数器溢出中断；
\$FFEE～\$FFFF	TIM2 通道 1 输入捕捉、输出比较中断；
\$FFF0～\$FFF1	TIM2 通道 0 输入捕捉、输出比较中断。

(8) 同步串行口 SPI 引起的中断。其中断和向量地址为

\$FFE8～\$FFE9	SPI 发送中断；
\$FFEA～\$FFEB	SPI 接受中断。

(9) 异步串行口 SCI 中断。其中断和向量地址为

\$FFE2～\$FFE3	SCI 发送中断；
\$FFE4～\$FFE5	SCI 接受中断；
\$FFE6～\$FFE7	SCI 发送错误。

(10) A/D 转换器中断(仅适用于 WAIT 模式，STOP 模式将关闭 A/D 转换器)。A/D转换结束将引起中断，向量地址为 \$FFDE～\$FFDF。

10.3 在 WAIT 和 STOP 模式下微控制器中各模块的工作状态

10.3.1 A/D 转换模块

在 WAIT 模式下 A/D 转换器继续工作，并且由 A/D 模块产生的中断请求将唤醒微控制器，如果程序中不希望 A/D 电路影响 WAIT 状态，可以在运行 WAIT 指令前，置位 ADC 控制寄存器中的 ADCH4～ADCH0 位，关闭 A/D 模块。运行 STOP 指令后 A/D 模块将被关闭，正在进行的 A/D 操作将暂停。外中断唤醒微控制器后，A/D 模块继续其操作。

10.3.2 断点模块(BRK)

在 WAIT 模式下 BRK 模块继续工作；STOP 方式下 BRK 被禁止，但 BRK 产生的中断请求将唤醒微控制器，同时将 BRK 状态寄存器中的 BW 位置 1。STOP 指令不影响 BRK 模块的寄存器状态。

10.3.3 中央处理器(CPU)

CPU 进入 WAIT 或 STOP 方式后将进行下列操作：首先清除状态寄存器中的中断屏蔽位 I，允许中断，然后关闭 CPU 时钟，停止操作。如果是中断将 CPU 唤醒，I 屏蔽位保持为 0；但如果是复位信号，I 将置 1，禁止中断。

10.3.4 时钟发生模块(CGM)

WAIT 模式下时钟发生模块保持工作状态。进入 WAIT 模式之前，为了进一步减小功耗，软件可以选择外部晶振而不选择 CGM 为系统时钟源(清除 PLL 控制寄存器 PCTL 中的 BCS 位)或干脆关闭 PLL 模块(清除 PLLON 位)。如果对功耗要求不很高，可以不关闭 PLL，这样微控制器唤醒后可迅速进入工作状态而不必等待时钟稳定。另外，如果系统设计为需要 PLL 中断唤醒，就不能关闭 PLL 电路。

在 STOP 模式下，如果系统配置寄存器中的 OSCSTOPEN 位为 0(缺省模式)，执行 STOP 指令将关闭 CGM 模块，并将与 CGM 模块有关的输出引脚(CGMXCLK，CGMOUT，CGMINT)置低。如果在 STOP 指令运行时，系统时钟 CGMOUT 是由锁相环压控振荡器时钟 CGMVCLK 二分频驱动的，系统将自动清除 PLL 控制寄存器 PCTL 中的 BCS 位，选择晶振作为系统时钟源。当从 STOP 状态下唤醒后，BCS 位保持不变(0)。也就是说，如果想重新启动 PLL，从 STOP 状态下恢复后，在程序中必须重新设置 PCTL。有关时钟发生模块的详细内容，请读者参考第 7 章。

10.3.5 看门狗模块(COP)

COP 模块在 WAIT 模式下保持工作状态。在 STOP 模式下，STOP 指令将关闭 COP 的时钟源 CGMXCLK，并清除 COP 的预分频器。系统配置寄存器中的 STOP 位允许 STOP 指令的执行。为了防止在程序中可能无意用 STOP 指令关闭 COP 功能，可以清除系统配置寄存器中的 STOP 位，禁止 STOP 指令。

10.3.6 外中断模块(IRQ)

IRQ 模块在 WAIT 和 STOP 状态下都保持工作状态。清除 IRQ 状态控制寄存器中的 IMASK1 位将使外中断唤醒微控制器。

10.3.7 键盘中断模块(KBI)

键盘中断模块在 WAIT 和 STOP 状态下都保持工作状态。清除键盘状态控制寄存器中的 IMASK1 位将使键盘中断唤醒微控制器。

10.3.8 低电压禁止模块(LVI)

如果在软件中打开低电压禁止功能，在 STOP 和 WAIT 模式下低电压禁止功能都将起作用，模块产生的复位信号将唤醒微控制器。

10.3.9 异步串行通信接口(SCI)

在 WAIT 模式下，异步串行通信接口继续工作。如果允许 SCI 中断，则接受或发送中断都将唤醒微控制器。出于降低功耗的考虑，在运行 WAIT 指令前，可以关闭 SCI 模块。

在 STOP 模式下，SCI 被关闭。STOP 指令不影响 SCI 的标志寄存器。由于 SCI 的工作时钟来自系统时钟，如果运行 STOP 指令时 SCI 正在接受或发送数据，则将停止操作，此时接受或发送的数据将会出错，所以应当避免这种情况。当微控制器被唤醒后，SCI 模块继续工作。

10.3.10 同步串行通信接口(SPI)

在 WAIT 模式下，同步串行通信接口继续工作。如果允许 SPI 中断，则 SPI 的接受或发送中断都将唤醒微控制器。也可以在运行 WAIT 指令前，关闭 SPI 模块进一步降低功耗。

在 STOP 模式下，SPI 被关闭。STOP 指令不影响 SPI 的标志寄存器。

10.4 低功耗应用举例

一般的微控制器应用开发项目对系统功耗的要求并不严格。但在某些特殊的场合，对于系统的整体功耗有近乎苛刻的要求。下面以目前十分流行的工业现场总线仪表为例，简要说明如何从微控制器软件编程、外围器件选择、电路设计等方面最大限度地减小系统功耗。

10.4.1 智能二线制仪表简介

现场总线是一种双向的、全数字的通信网络，主要用于过程自动化和制造自动化底层的现场设备互连。现场总线将各种仪表和执行设备连接起来，由计算机控制，实现生产过程控制的自动化。由于要应用于各种不同的生产现场，所以现场总线标准的基本要求是安全和可靠。在石油、化工等生产部门中，生产过程现场往往含有多种易燃易爆气体，它们与空气一起形成爆炸性混合物，这样的过程现场是危险场所。对在危险场所中使用的控制设备必须采取防爆措施。

过程控制装置使用的防爆结构主要有隔离型和本质安全型(安全火花型)。隔离型防爆结构是将电路和接线端子全部封装在隔爆表壳内，表壳强度要足够大，接合面间隙要足够深，接合面宽度要足够窄。这样，即使装置产生火花和爆炸时，也不会引起表壳外的爆炸性混合物爆炸。这类防爆结构不能用于危险程度高的场所中，在运行中不能进行检修和调校，且会因磨损、老化而逐渐丧失防爆能力。

本质安全型防爆结构是指在正常状态或事故状态下所产生的火花及达到的温度均不能引起爆炸性混合物爆炸，这是由具体的装置设计所保证的。所以，它要求仪表采用尽量低的工作电压和工作电流。设备的功耗越低就越不容易产生火花，也就越安全。这种结构原理上可适用所有的危险场所，可在运行状态下进行检修和调整，其安全性能也不会随时间而变化，是理想的解决方案。

为了达到安全标准，目前一种十分流行的现场总线采用 4～20mA 标准，简单地说就是把一个 0/4～20mA 的直流电流通过双线(分别用于电流的流入和流出)送给仪表或执行器，执行器能根据此电流的大小到达预定的控制目标。例如，用调节阀做执行器时，希望 0/4mA 代表阀门全关，20mA 代表全开，或者相反；当然也可以按照其他特定的阀位-电流曲线工作。所谓二线制仪表，是指电源和信号线公用两根导线的仪表，即不增加额外的电源，仪表的供电完全是从控制信号中取出的。二线制仪表的执行器件可以采用很多非电力的执行设备，例如压缩空气动力的阀门和开关，可以用压电陶瓷控制。采用气动部件可以进一步提高设备在特殊应用环境中的安全性。

由此可见，二线制仪表由于电源本身取自信号线，设备工作在毫安级，功耗很小，在工作中不会产生电火花。所以在构成本质安全的防爆结构时，具有很大的优势和方便条件。

10.4.2 智能二线制仪表的硬件电路

由上所述可知，整个硬件设计的难点在于二线制系统对功耗的要求苛刻，整个系统（包括压电控制阀的驱动）都应能通过 4～20mA 信号线供电。因此硬件设计主要是围绕这一方面进行的。

系统所允许的最大功耗可以这样估算：若允许输入电流有 10％的波动，信号线上的最小电流实际上只有 3.6mA。假定在输入端能取出 10V 电压的话，则整个系统的功耗不应大于 36mW。如此低的功耗，要求在器件选择上和电路设计上多下功夫。在电路设计上，应尽量减少不必要的功率消耗，提高各部分的供电效率；在电路的实现上，采取了选用低功耗或微功耗器件、降低器件的工作电压以及降低微控制器工作电压和频率等具体措施。另一方面，必须认识到，低的电压和功耗与系统的抗干扰性、精度、速度等性能存在着固有的矛盾，在降低功耗的同时必然意味着其他方面的性能损失。

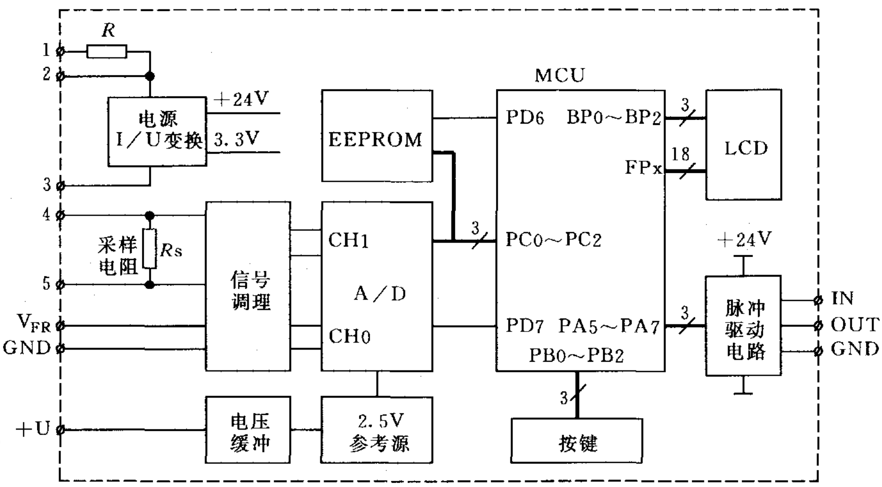
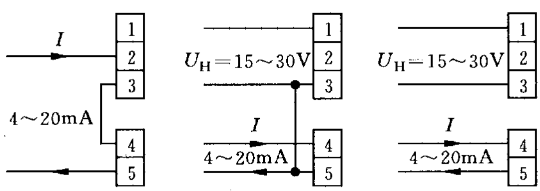


图 10-4 电路原理框图

图 10-4 左侧的接线端 1～5 为一组，是电源和设定信号入口。根据不同的实际需要，智能定位器可以有两线、三线和四线制等几种不同的接法，如图 10-5。两线制接法只需外加 4～20mA 电流即可；而三线和四线制除了电流信号外，还需外接 15～30V 电压作为额外的电源，二者的区别在于电流信号与电源电压是否共地。



(a) 二线制　(b) 三线制　(c) 四线制

图 10-5 智能定位器的 3 种接线方式

下面简单介绍图 10-4 中的主要功能模块。

(1) 电源变换模块。该模块的任务是把输入的电流(二线制)或电压(三线、四线制)变换成 3.3V 和 24V 两组电源,供微控制器和其他器件使用。实现方法是通过振荡升压,将低电压变换成系统需要的电压。对电源变换模块的要求是高效,自身消耗的功率要小。

(2) 信号调理和 A/D 变换电路。它主要由运放和 A/D 变换器构成。所谓信号调理,就是将输入信号的幅值等特性转换成统一的形式,以供系统处理。例如将电流大小转换成表示阀门开关大小的数字量。

(3) 压电陶瓷阀驱动电路和驱动电平。驱动电路的功能主要是将来自微控制器的幅度为 3.3V 的驱动控制信号(连续的或者脉冲的)转换成幅值为 24V 的驱动脉冲来推动压电陶瓷气动阀动作。

10.4.3 智能二线制仪表的低功耗措施

1. 微控制器

为降低功耗,在微控制器的功耗和速度之间取了某种折衷方案。把微控制器的总线频率降低到 1MHz,同时把它的工作电压降低到 3.3V。加之在软件设计时使用了节电的 WAIT 方式,如果处于 WAIT 方式的时间平均占到 50%,则平均功耗还可再降低一半。

除了努力降低微控制器的功耗外,在选择外围器件上也要仔细计算功耗,选择微功耗器件。

2. 微功耗串行 ADC LTC1594L

LTC1594L 是 Linear Technology 公司的 4 通道 12 位微功耗串行 ADC。其转换方式为开关电容、逐次渐进式转换,带有采样保持。LTC1594L 工作电压可低至 2.7V,在转换状态时供电电流典型值为 0.16mA,空闲状态时仅为 1nA。LTC1594L 用在 A/D 变换电路。

3. 低压差线性稳压器 LT1521

LT1521-3.3 是 Linear Technology 公司的一种低压差、微功耗线性稳压器。它可以在稳压压差 0.5V 时提供 300mA 的输出电流。它的工作状态电流为 12μA,备用状态下为 6μA,并且耗电电流不会随着压差的增大而改变,因此在许多电池供电和低功耗系统中得到应用。在本例中 LT1521 用于电源变换模块。

4. 微功耗双运放 LT1178

LT1178 是 Linear Technology 公司的产品。它在保持微功耗的同时保持了较高的精度,在电压为 5V、室温时消耗电流典型值为 17μA,失调电压为 30μV,失调电流为 50pA。LT1178 可以在低至 2V 的单电源下工作。

5. 1kB 串行 EEPROM 93LC46A

EEPROM 选用了 Microchip 公司的 93LC46A，属于低功耗产品。其结构形式为 128 * 8 位，工作电压可低至 2.5V，在 3V、1MHz 下读出电流最大为 500μA，写入电流为 1.5mA，单字节写入时间最长为 6ms。

需要注意的是，93LC46A 的读、写时序与 Motorola 系列微控制器的 SPI 接口并不完全兼容，而另一种产品 AT25010 可以做到完全兼容。之所以采用前者，主要还是因为其功耗只有后者的一半。在这种情况下，为了满足低功耗的要求，对于时序中的不兼容部分(读出时序部分)，可以不用 SPI，直接通过指令在相应的 I/O 口线上产生所需要的时序。

6. 微功耗 2.5V 基准源 LT1004-2.5

LT1004-2.5 是 Linear Technology 公司的产品。它的特点是微功耗、高精度、易于使用。通常状况下能保证在低达 10μA 的电流下正常工作，初始精度为±20mV。LT1004 可以直接替换 LM285 系列，同时提供更高的精度和更低的功耗。

10.4.4 智能二线制仪表的软件

智能定位器系统软件设计的主要任务是实现人机交互接口，如 LCD 显示和键盘、菜单选择参数等，实现自我配置功能，实现参数的断电保存，实现各种功能的计算和修正，以及实现基本控制算法和脉冲转换等。在软件设计时，首先实现了一个简单的基于时钟中断的实时内核，在此基础上，把要完成的功能编制成不同的任务模块，根据各个任务所要求的执行频率的不同进行统一调度，进而完成全部系统功能。在软件的编制过程中，使用了微控制器的 WAIT 低功耗节电方式来降低系统功耗。

1. 基于时间的实时内核

所谓实时内核，实际上非常类似于一个简单的操作系统。它提供了较强的灵活性和快速的软件开发速度。不同的任务模块可以被排进内核的任务表中，通过内核的统一调度得以执行，并且新的任务模块也可以很容易地加入到任务表中而不会影响到先前排定的任务。

2. 微控制器采用实时内核及 WAIT 模式

传统的微控制器程序通常都是基于单任务的，这种机制具有简单直观、易于控制的优点。然而由于程序只能按单一线索顺序执行，缺乏灵活性，在复杂系统中难以胜任。而实时内核的使用，使各个任务可以在各自的时间片内轮流运行，从而形成宏观上同时运行的多任务效果。可以说把多任务机制带进了微控制器中。多任务方式下的调度方式如图 10-6 所示。在多任务机制下，各功能模块由系统调度程序启动，执行完毕后返回调度程序，彼此处于同等地位。其执行顺序比较灵活，且在执行过程中可以动态改变。各个任务

的执行频率要求表现为它们的优先级。适当安排各任务模块的执行频率，可以自然地满足系统内各种复杂的定时要求。当所有的任务都结束后，系统进入 WAIT 模式，降低功耗，直到有外部事件唤醒微控制器。

在这种机制下，程序的调试和维护也比较容易。因为各个程序模块功能相对独立，相互影响小，可以方便地分别单独调试，找出问题所在。对程序的扩充也不必改动程序的原有结构。

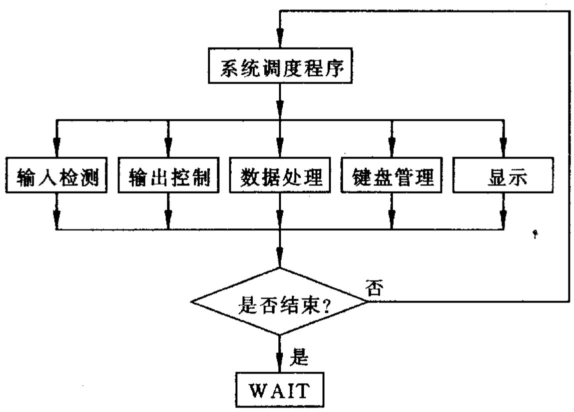


图 10-6　多任务方式下的调度方式

10.4.5　智能二线制仪表系统功耗的估算

系统功耗包括

(1) 微控制器在对程序代码进行必要的分析后，预计 CPU 平均时间占用率低于 50%。因此，考虑到在编程中使用了 WAIT 方式，微控制器供电电流估算为 1.2mA，消耗功率约 4mW。

(2) 运放 LT1078 及其相关电路。运放本身功耗为 0.15mW，相关电路功耗为 0.15mW，总共约 0.3mW。

(3) AD 转换及参考源电路。LT1594L 在空闲状态下功率消耗可以忽略，转换状态下为 0.5mW，转换一次的时间为 400μs，每 12.5ms 转换两次，因此认为 ADC 有 10% 的时间在工作，功耗为 0.05mW；参考源因为要供给外部反馈电位器，电流约为 0.5mA，所以功耗约为 1.7mW。因此这部分功耗估算为 1.8mW。

(4) EEPROM 电路。EEPROM 需要读写的时间极少，且只在系统上电复位或在配置方式下进行，所以认为 93LC46A 工作在空闲方式下，功耗低于 0.03mW。

(5) LCD 电路。消耗功率估算为 0.1mW。

(6) 压电阀驱动电路。压电阀功耗极小，暂不计，只有驱动电路消耗功率，估算为 2mW。

以上功耗不计电源转换电路共约 8.3mW，加上其他损耗后也不会超过 10mW，这是

电源转换电路在最小工作电流下完全可以承受的。以上指标表明，本例中所设计的电路不但达到了二线制供电的基本要求，而且还留有一定的余量。若对电源电路进行必要的改进，例如改进电路设计，选用损耗小、特性好的变压器磁心等措施后，相信系统还可以在更低的电流下工作。这就是说，完全可以在此基础上扩充系统功能，并且可能使速度更快、功能更强的微控制器的引入成为可能。

第 11 章　监控 ROM 程序

11.1　监控 ROM 程序的功能

MC68HC08 系列的微控制器将 MC68HC05 系列的 EPROM 更换成了 Flash，使得程序的写入和更改更加方便快捷，这是 MC68HC08 的一大特点。为了配合 Flash 的使用，发挥 Flash 快速擦写的优点，Motorola 公司为 MC86HC08 系列的微控制器提供了监控 ROM。监控 ROM 是指微控制器出厂前固化在芯片内部 ROM 中的一段程序，其中包含了有关系统检测、Flash 编程以及串行通信等功能的代码。这就使得微控制器多了一种不同于正常用户方式的特殊操作方式，称为监控方式。在特定条件下，微控制器可以不进入正常的用户方式，而是进入到监控方式中，这时从 $FEFE、$FEFF 处取得复位向量，执行内部 ROM 中固化的代码。Motorola 公司提供的编程器和仿真器都会使用监控 ROM 中的代码。

使用监控方式执行监控 ROM 的代码，可以使未经用户编程的芯片执行特定的功能。监控 ROM 可以通过单一的一条信号线与主机进行串行通信，接收和执行预先定义的主机命令，如读写内存、执行程序等，并返回结果。适当运用监控方式和这些主机命令，能够完成一些特殊功能，例如：

(1) 下装代码到 RAM 或 Flash 内存中；

(2) 执行 RAM 或 Flash 中的程序代码；

(3) Flash 内存的加密；

(4) Flash 内存擦除/写入/校验；

(5) 与主计算机进行标准的不归零传号/空号串行通信，波特率为 4800b～28.8kb；

(6) 在线编程；

(7) 用户方式 Flash 编程。

除了这些通用功能外，不同型号芯片的监控 ROM 根据芯片组成模块的不同，往往还提供一些只应用于本型号芯片的特殊功能代码。例如包含 USB 模块的芯片，其监控 ROM 就包含了一个完整的 USB 设备所需的代码，当进入监控方式的时候，PC 机将把该芯片识别为一个标准的 USB 设备。

对于不同的微控制器，其监控 ROM 的位置、大小和内容通常都是不同的。这给解读和应用监控 ROM 程序带来了很大的麻烦。本教材主要以指定的两种芯片 MC68HC908GP32 和 MC68HC908JL3 为例来探讨监控 ROM。这两款芯片也是 Motorola 公司的 MC68HC08 系列微控制器的代表产品，今后会得到很大的推广。读懂这两款芯片的监控 ROM 会对今后的应用开发带来很大的方便。因为监控 ROM 的代码是固定的，其中的各个子程序对入口参数的要求非常严格，并且对 RAM 区域的使用也是固定的，因此用户使用监控 ROM 的代码需要非常小心，不能弄错子程序的入口代码，也不能

使用监控 ROM 已经使用的 RAM 区域。Motorola 公司随时可能更改监控 ROM 代码的内容，读者需要根据实际使用的芯片来确定监控 ROM 的内容。

MC68HC908GP32 的内部存储器包括 32kB 的 Flash、512B 的 RAM。这个容量是比较大的，因此 MC68HC908GP32 更多地使用 RAM 来存储从上位计算机得到的监控代码，而实际固化的代码非常少。如果用户使用 Motorola 公司的 MC68HC08 串行编程板和 ICS08GP20 目标板来开发 MC68HC908GP32，会发现在进入编程环境时需要下载一个“908_GP32. 08P”文件。这个文件实际包含了擦除、写入 Flash 的子程序代码。所以这里主要讨论 MC68HC908JL3 芯片的监控 ROM。同 MC68HC908JL3 监控 ROM 相同或相近的其他型号的芯片还有：MC68HC908GR8、MC68HC908KX8、MC68HC908JK3 和 MC68HC908JB8。如果读者使用到上述的芯片，也可以参考本章来了解其片内的监控 ROM。

MC68HC908JL3 的片内资源比 MC68HC908GP32 要少得多，只有 128B 的 RAM 和 4kB 的 Flash，所以就不能像 MC68HC908GP32 那样使用下载监控程序到 RAM 中的编程方法。因此，Motorola 公司将大部分的程序固化到监控 ROM。在地址范围 $FC00～$FDFF 和 $FE10～$FFCF 处，有 960B 的监控和自检程序。这些程序包括使用半双工口的通信子程序、Flash 的擦除、编程和校验程序等。

因为 MC68HC908JL3 的片内 Flash 只有 4kB，容量很小，如果能够直接使用监控 ROM 的程序代码，必然可以大大节约程序容量。同时，如果希望实现用户程序本身对 Flash 的更改，也就是在线编程，也只有使用监控 ROM 程序。

11.2 监控 ROM 程序详解

利用 Motorola 公司提供的 MC68HC908GP20 和 MC68HC908GP32 的集成调试工具，可以将 MC68HC908JL3 监控 ROM 的机器码读出，并进行反汇编。下面分析反汇编得到的代码。

在反汇编的代码中，首先看到的就是一串跳转语句，如下所示：

地址	机器码	反汇编码
FC00	CCFC24	JMP FC24
FC03	CCFC38	JMP FC38
FC06	CCFD21	JMP FD21
FC09	CCFC74	JMP FC74
FC0C	CCFD08	JMP FD08
FC0F	CCFD72	JMP FD72
FC12	CCFDBD	JMP FDBD
FC15	CCFFB6	JMP FFB6
FC18	CCFDDD	JMP FDDD
FC1B	CCFDE9	JMP FDE9
FC1E	CCFFA1	JMP FFA1
FC21	CCFF40	JMP FF40

对于这些跳转语句，可以认为是子程序的跳转入口。各个子程序的入口地址为"JMP"语句所指向的跳转地址，这部分的子程序在今后的不同版本的芯片中可能有所不同，但使用时调用的方法是固定的。使用"JSR 'JMP 语句的地址' "就可以了。

JMP 语句的跳转地址对应不同的子程序，通过解读这些子程序，可以了解监控方式的工作过程，也为成功地调用子程序做好准备。根据解读的子程序的功能，可以给这些子程序命名，并且使用 EQU 的伪指令将这些子程序名与实际的子程序地址连接起来。例如：

```
GETBYTE     EQU     $FC00
RDVRRNG     EQU     $FC03
ERARNGE     EQU     $FC06
PRGRNGE     EQU     $FC09
SUB_DELAY   EQU     $FC0C
```

这样，用户在编程时，只要按照"JSR GETBYTE"这样的格式就可以调用监控 ROM 的子程序了。

下面解释一些比较重要和可能使用的子程序，包括各个子程序的功能以及它们的入口参数、出口参数和注意事项。

1. 接受一个字节的子程序 GETBYTE

MC68HC08 系列芯片使用了一个通用 I/O 口，通过一定的外围电路构成一个半双工的串行通信口。监控方式就是通过这个通信口来实现对 Flash 的读写擦除及其他功能。不同的芯片，这个 I/O 口也不同，并且通信的波特率也需要参考这个芯片的具体说明。以 MC68HC908JL3 为例，它使用 PB0 口构成通信口，通信方式采用标准不归零方式、9600 波特率、一个起始位、没有停止位和检验位。

这个子程序不需要入口参数，接受完成后把接受到的字节保存在累加寄存器 A 中。需要注意的是：

(1) 一旦调用这个子程序，在没有接受到有效的字节前，该子程序将一直占用 CPU；

(2) 该子程序忽略起始位；

(3) 该子程序对结束位不做任何校验。

下面介绍这个子程序的反汇编代码。

地址	机器码	反汇编码	解读注释
FC24	0001FD	BRSET 0,PORTB,FC24	;等待起始位
FC27	CDFECD	JSR RCV_BIT_TO_C	= FECD
FC2A	25F8	BCS FC24	;检验起始位，继续等待
FC2C	A680	LDA #80	
FC2E	CDFECD	JSR RCV_BIT_TO_C	;接收一个位
FC31	46	RORA	
FC32	24FA	BCC FC2E	;判断是否已经接收完一个字节
FC34	CDFECD	JSR RCV_BIT_TO_C	;接受结束位，不做检验

FC37	81	RTS

在这个子程序中，又调用了 RCV_BIT_TO_C 子程序。RCV_BIT_TO_C 子程序在其他的子程序中还有调用，下面简单介绍一下这个子程序。

RCV_BIT_TO_C 子程序通过半双工的通信口接受一个位，并且把这个位保存在条件寄存器(CCR)的进位位 C 位后返回。RCV_BIT_TO_C 子程序的反汇编码为：

地址	机器码	反汇编码	解读注释
FECD	89	PSHX	
FECE	87	PSHA	
FECF	A610	LDA #10	
FED1	AE17	LDX #17	
FED3	010100	BRCLR 0,PORTB,FED6	;C 位根据 PB0 的电平改变
			;PB0 为高，C 位为 1
FED6	A200	SBC #0	;(A) = (A)-(0)-(C)，此时累加
			;器第 0 位和 C 位相同
FED8	5BF9	DBNZX FED3	;延迟一个位的时间
FEDA	49	ROLA	;将累加器第 0 位送到 C 位
FEDB	86	PULA	
FEDC	88	PULX	
FEDD	81	RTS	

2. 擦除指定地址的 Flash 内容的子程序 ERARNGE

擦除 Flash 比写入 Flash 要简单得多。对 Flash 的擦除操作只有两种形式：页擦除和整体擦除。页擦除是擦除指定的 Flash 地址所在的页(参见第 5 章)，即是 $xx00～$xx7F 或 $xx80～$xxFF 的一段地址，而整体擦除是擦除所有的 Flash 区域。这两种方式在入口参数中指明。

该子程序的入口参数为

(1) RAM 地址 $0088 中第 6 位指定擦除方式，如果该位为 1，就是整体擦除。

(2) H:X 寄存器中存放需要擦除的 Flash 地址，该地址并不需要是起始地址，如果是整体擦除，该地址可以是 Flash 区域的任意地址；如果是页擦除，该地址只要包含在即将被擦除的 Flash 页中即可。

该子程序无出口参数。

需要注意的是，存放向量的 Flash 区域只能进行整体擦除。也就是说，即使只更改了一个向量，也要对整个向量区重新编程。

下面是这个子程序的反汇编码。

地址	机器码	反汇编码	解读注释
FD21	9B	SEI	
FD22	8B	PSHH	
FD23	89	PSHX	

```
FD24    4F        CLRA
FD25    AA02      ORA  #2              ;置擦除位
FD27    0D8802    BRCLR 6,0088,FD2C    ;整体擦除?
FD2A    AA04      ORA  #4
FD2C    C7FE08    STA FLCR
FD2F    C6FE09    LDA FLBPR            ;Flash 保护寄存器
FD32    F7        STA ,X               ;写入任意地址
FD33    450001    LDHX #1
FD36    B689      LDA 0089
FD38    ADCE      BSR SUB_DELAY        ;调用延时程序
FD3A    45FE08    LDHX #0FE08
FD3D    F6        LDA ,X
FD3E    AA08      ORA  #8              ;打开 Flash 高压
FD40    F7        STA ,X
FD41    0D8805    BRCLR 6,0088,FD49    ;不是整体擦除
FD44    4500A6    LDHX #0A6
FD47    2003      BRA FD4C             ;整体擦除,延时 4ms
FD49    450029    LDHX #29
FD4C    B689      LDA 0089             ;页擦除,延时 1ms
FD4E    ADB8      BSR SUB_DELAY
FD50    45FE08    LDHX #0FE08
FD53    F6        LDA ,X
FD54    A802      EOR  #2              ;清除"擦除"位
FD56    A4FB      AND  #0FB            ;清除"整体擦除"位
FD58    F7        STA ,X
FD59    0D8805    BRCLR 6,0088,FD61
FD5C    450004    LDHX #4
FD5F    2003      BRA FD64
FD61    450001    LDHX #1
FD64    B689      LDA 0089
FD66    ADA0      BSR SUB_DELAY
FD68    45FE08    LDHX #0FE08
FD6B    F6        LDA ,X
FD6C    A808      EOR  #8              ;关闭 Flash 高压
FD6E    F7        STA ,X
FD6F    88        PULX
FD70    8A        PULH
FD71    81        RTS
```

3. Flash 编程子程序 PRGRNGE

该子程序完成对 Flash 的编程,它将 RAM 区域中的数据队列的内容写入指定的

Flash 地址。这最能体现 MC68HC08 系列微控制器 Flash 存储器的优点。基本上每一款芯片的监控 ROM 都会包含实现这个功能的子程序。

入口参数为

(1) H:X 寄存器存放需要编程的 Flash 的起始地址。

(2) RAM 地址 $ 008A: $ 008B 存放需要编程的 Flash 的结束地址。

(3) 需要编程的内容放在从 RAM 地址 $ 008C 开始的区域里。

无出口参数。

需要注意的是，由于这个子程序要改动 Flash 的内容，必须在调用之前修改 Flash 保护寄存器，取消对即将编程的 Flash 区域的保护；这个子程序本身并没有对将要进行编程操作的 Flash 区域是否空白进行检查，也没有在编程结束以后对编程的结果进行校验，所以在实际使用的时候，还要调用另外的子程序来完成这些工作。例如调用前面讲到过的 ERARNGE 来进行擦除工作，调用后面将讲到的 RDVRRNG 来进行校验。在这个子程序中，还调用了一个延时的程序 SUB_DELAY。后面将详细介绍这个子程序。

另外需要特别强调的是，在这个子程序写入 Flash 的时候，是将 RAM 区域的对应字节的内容写入到 Flash 中。因此，调用者必须保证 RAM 区的数据队列的长度和指定的 Flash 区域的起始、结束地址是统一的。如果两者不一致，该子程序将有可能出现编程错误。

下面是这个子程序的反汇编码。

地址	机器码	反汇编码	解读注释
FC74	9B	SEI	
FC75	4F	CLRA	
FC76	87	PSHA	;子程序内部变量
FC77	89	PSHX	
FC78	8B	PSHH	
FC79	9F	TXA	
FC7A	AE20	LDX #20	
FC7C	8C	CLRH	
FC7D	52	DIV	;A←商，H←余数
FC7E	8B	PSHH	
FC7F	9F	TXA	
FC80	9EE001	SUB 1,SP	;减去余数
FC83	8A	PULH	
FC84	8A	PULH	
FC85	88	PULX	;编程起始地址
FC86	87	PSHA	;这里的 A= $ 20-余数
WRITE_FLH:			
FC87	A601	LDA #1	
FC89	CAFE08	ORA FLCR	;置位编程位
FC8C	A4F9	AND #0F9	;清除“块擦除”位

```
FC8E    C7FE08    STA FL
FC91    C6FE09    LDA FLBP
FC94    F7        STA ,X              ;写入任意地址,任意数据
FC95    8B        PSHH
FC96    89        PSHX
FC97    450001    LDHX #1
FC9A    B689      LDA 0089            ;调用延时程序
FC9C    AD6A      BSR SUB_DELAY
FC9E    45FE08    LDHX #0FE08
FCA1    F6        LDA ,X              ;打开编程电压
FCA2    AA08      ORA #8
FCA4    F7        STA ,X
FCA5    450001    LDHX #1
FCA8    B689      LDA 0089            ;调用延时程序
FCAA    AD5C      BSR SUB_DELAY
FCAC    88        PULX
FCAD    8A        PULH
WRITE_FLH_LINE:
FCAE    8B        PSHH
FCAF    89        PSHX
FCB0    8C        CLRH
FCB1    9EEE04    LDX 4,SP
FCB4    E68C      LDA 8C,X            ;读取需要编程的字节
FCB6    88        PULX
FCB7    8A        PULH                ;指定编程地址
FCB8    F7        STA ,X              ;写入数据
FCB9    8B        PSHH
FCBA    89        PSHX
FCBB    450002    LDHX #2
FCBE    B689      LDA 0089
FCC0    AD46      BSR SUB_DELAY       ;调用延时程序
FCC2    88        PULX
FCC3    8A        PULH
FCC4    AF01      AIX #1              ;编程地址指向下一个地址
FCC6    9E6C02    INC 2,SP            ;行地址递减
FCC9    9E6A01    DEC 1,SP
FCCC    758A      CPHX 008A
FCCE    2205      BHI FCD5            ;判断是否已经到达结束地址
FCD0    9E6D01    TST 1,SP            ;判断是否写完一行
FCD3    26D9      BNE WRITE_FLH_LINE
FCD5    8B        PSHH
FCD6    89        PSHX
```

```
FCD7    45FE08    LDHX #0FE08
FCDA    F6        LDA ,X
FCDB    A801      EOR #1
FCDD    F7        STA ,X             ;清除“编程”位
FCDE    450001    LDHX #1
FCE1    B689      LDA 0089
FCE3    AD23      BSR SUB_DELAY      ;调用延时子程序
FCE5    45FE08    LDHX #0FE08
FCE8    F6        LDA ,X
FCE9    A808      EOR #8
FCEB    F7        STA ,X             ;关闭编程电压
FCEC    86        PULA
FCED    8A        PULH
FCEE    9EEB01    ADD 1,SP
FCF1    97        TAX
FCF2    8B        PSHH
FCF3    86        PULA
FCF4    A900      ADC #0
FCF6    87        PSHA
FCF7    8A        PULH
FCF8    A620      LDA #20            ;开始另一行的编程
FCFA    9EE701    STA 1,SP
FCFD    AFFF      AIX #-1
FCFF    758A      CPHX 008A          ;判断是否到达结束地址
FD01    AF01      AIX #1
FD03    2582      BCS WRITE_FLH
FD05    86        PULA
FD06    86        PULA
FD07    81        RTS
```

4. Flash 校验的子程序 RDVRRNG

这个子程序完成写入 Flash 后的检验。它有两个主要的功能：读取一段 Flash 存储区；将指定地址区域的 Flash 内容和 RAM 中的指定内容进行比较。

实际上每次调用这个子程序，这两个功能都执行了，并返回指定地址的 Flash 存储区内容。这个子程序的灵活性在于调用时可以指定是由监控 ROM 本身还是由上位计算机的配套软件进行读取字节的比较。如果调用该子程序时累加寄存器 A 为零，那么读取的数据将直接送到通信口向上位机发送。如果累加寄存器 A 不为零，那么读取的数据将和 RAM 区对应数据进行比较，如果不同，这个数据将保存在对应的 RAM 区中，覆盖原来的内容。读取并比较的 Flash 区域的起始和结束地址都作为参数在调用这个子程序时传入。如果读取的 Flash 内容和 RAM 数据队列中对应的内容全部吻合，条件寄存器 CCR

的进位位 C 位将置位，这个子程序还将检查所有返回数据的校验和，并通过累加寄存器 A 传递出来。

入口参数为

(1) 累加器 A，A 为零，读取的数据发送到上位机；A 不为零，读取的数据保存在 RAM 中。

(2) H:X 寄存器存放读取的 Flash 区域的起始地址。

(3) RAM 地址 $ 008A：$ 008B 中存放读取的 Flash 区域的结束地址。

(4) 从 RAM 地址 $ 008C 开始存放等待比较的目标字节。

如前所述，出口参数包括累加寄存器中所有读取数据的校验和和条件寄存器的进位位。

下面是这个子程序的反汇编码。

地址	机器码	反汇编码	解读注释
FC38	87	PSHA	;使用堆栈作为内部变量的使用空间
			;该字节命名为 STACK1，保存参数 A
FC39	4F	CLRA	
FC3A	87	PSHA	;命名为 STACK2，现在为 0
FC3B	87	PSHA	;命名为 STACK3，现在为 0
FC3C	43	COMA	
FC3D	87	PSHA	;命名为 STACK4，现在为 FF
FC3E	F6	LDA ,X	
FC3F	9E6D04	TST 4,SP	;检验入口的 A(STACK1)是否为零
FC42	2715	BEQ FC59	
FC44	89	PSHX	;命名为 STACK5
FC45	8B	PSHH	;命名为 STACK6，保存起始地址
FC46	9EEE04	LDX 4,SP	;STACK3
FC49	8C	CLRH	
FC4A	E18C	CMP 8C,X	;与目标字节比较
FC4C	2707	BEQ FC55	
FC4E	E78C	STA 8C,X	;不同，覆盖原来内容
FC50	AE7E	LDX #7E	
FC52	9EEF03	STX 3,SP	;STACK4
FC55	8A	PULH	
FC56	88	PULX	
FC57	2003	BRA FC5C	
FC59	CDFEA9	JSR SENT_BYTE	;入口 A=0，直接将读取的数据发送
FC5C	9EEB03	ADD 3,SP	;STACK2
FC5F	9EE703	STA 3,SP	;STACK2，存放检验和
FC62	9E6C02	INC 2,SP	;STACK3
FC65	758A	CPHX 008A	;判断是否到达结束地址
FC67	2404	BCC FC6D	

FC69	AF01	AIX #1	
FC6B	20D1	BRA FC3E	;继续读取并比较下一个字节
FC6D	86	PULA	
FC6E	84	TAP	;设置条件寄存器 CCR
FC6F	86	PULA	
FC70	86	PULA	
FC71	A701	AIS #1	
FC73	81	RTS	

在这个子程序中，又调用了 SEND_BYTE 子程序，这个子程序用来向串行口发送一个字节的数据，需要发送的数据通过累加器传入。

下面是 SEND_BYTE 的反汇编码。

地址	机器码	反汇编码	解读注释
FEA9	89	PSHX	
FEAA	87	PSHA	
FEAB	A60A	LDA #0A	;准备好起始位和结束位
FEAD	0101FD	BRCLR 0,PORTB,FEAD	;等待上一次传输结束
FEB0	AEA9	LDX #0A9	
FEB2	5BFE	DBNZX FEB2	;等待两个位的时间
FEB4	99	SEC	;起始位
FEB5	2009	BRA FEC0	
FEB7	9E6601	ROR 1,SP	;发送数据位
FEBA	2404	BCC FEC0	
FEBC	1105	BCLR 0,0005	
FEBE	2004	BRA FEC4	
FEC0	1005	BSET 0,0005	;根据数据位决定通信口电平
FEC2	2000	BRA FEC4	
FEC4	AE4F	LDX #4F	
FEC6	5BFE	DBNZX FEC6	;等待一个比特
FEC8	4BED	DBNZA FEB7	;继续发送
FECA	86	PULA	
FECB	88	PULX	
FECC	81	RTS	

5. 延迟子程序 SUB_DELAY

该子程序的功能是延时，在进行 Flash 的编程、擦除等操作的时候，往往需要进行精确的延时。这个子程序的延时时间可以通过入口参数来调整，以适应不同的延时需求。

入口参数为 A 和 X：延时的时间。

延时的计算公式为(单位为内部总线周期)：((A) * 6+19) * (X)。

下面是 SUB_DELAY 的反汇编码。

地址	机器码	反汇编码	解读注释
FD08	A004	SUB #4	
FD0A	650000	CPHX #0	
FD0D	2711	BEQ FD20	
FD0F	87	PSHA	
FD10	C7FFFF	STA FFFF	
FD13	9D	NOP	
FD14	410004	CBEQA #0,FD1B	
FD17	21FE	BRN FD17	
FD19	4BFC	DBNZA FD17	
FD1B	AFFF	AIX #−1	
FD1D	86	PULA	
FD1E	20EA	BRA FD0A	
FD20	81	RTS	

上面详细介绍了 MC68HC08 系列微控制器监控 ROM 的几个主要的子程序，在不同型号的微控制器的监控 ROM 中还可能包含一些其他功能的子程序，例如在 MC68HC908JL3 的监控 ROM 中包含了判断一个字节是否为 ASCII 码的子程序。但是这些子程序主要提供给 Motorola 的仿真器和编程器的软件使用，功能比较简单，实现起来比较容易，这里就不再讲述了。

掌握本章介绍的这几个子程序，对于理解 Flash 的读写原理，正确高效地使用 MC68HC08 系列的 Flash 的特性都会有很大的帮助。

第12章　电磁兼容性

随着电子技术的发展，微控制器的应用越来越广泛，微控制器的主频也越来越高，伴随而来的电磁兼容问题也越来越值得重视。一个由微控制器构成的系统，能不能可靠地工作，能不能达到电磁兼容的标准，与其制造工艺、印刷线路板的设计与布线有很大关系。印刷线路板的好坏往往直接影响系统的可靠性与抗干扰能力。

关于电磁兼容的理论，如信号的传输、交叉干扰等方面是非常成熟的。在相应的电子学著作中有详尽的理论推导和经验公式。高级的计算机辅助设计软件能进行印刷板一级的噪声模拟与分析。本章不打算涉足这方面的理论，仅从实用的角度介绍一些电磁兼容的基本概念以及设计一个基于微控制器的系统要注意些什么。这些内容在我国大学教材中很少出现，加入这一章的目的，是希望读者在了解微控制器原理与应用基础上还能了解设计与制造中的工艺问题。

12.1　电磁兼容的基本概念

国际电工技术委员会（IEC）对电磁兼容的定义为：电磁兼容是一种能力，它在其电磁环境中能完成它的功能，而不至于在其环境中产生不能容忍的干扰。可见从电磁兼容的观点出发，除了要求设备能按设计要求完成其功能外，还有下列2点要求。

（1）系统本身抗电磁干扰能力要强，不易受到外界环境的干扰。

（2）系统本身不应成为一个噪声源，产生对其他仪器、设备的电磁干扰。

电磁兼容中最重要的概念是电磁干扰，解决电磁兼容问题就是抑制电磁干扰的过程。由干扰源发出干扰能量，经过耦合途径将干扰能量传输到敏感设备，使敏感设备的工作受到影响，这一过程称为电磁干扰效应。因此，形成电磁干扰必须具备下列3个基本要素。

（1）电磁干扰源。它指产生电磁干扰的任何元器件、设备或自然现象；

（2）耦合途径。它指将电磁干扰能量传输到受干扰设备的通路或媒介；

（3）敏感设备。它指受到电磁干扰影响，或者说对电磁干扰发生响应的设备。

由此可见，抑制电磁干扰就是要控制产生电磁干扰的3个要素，对其中任何一个因素的有效控制，都会增强系统的电磁兼容能力。

12.2　电磁兼容组织与标准

由于有用信号与干扰信号都是遵循物理定律和电磁波的传播规律向空间传播的，它所产生的影响不受国界和其他政治区域的限制，因此为了实现电磁兼容，各种世界性组织和地区性组织进行国际间的协调行动是非常重要的。例如，在全世界范围内对人为干扰提出规定性限制，规定一些标准与规范，对一些固定业务进行合理的保护等，这对于推动

电磁兼容技术的发展，实现系统的电磁兼容是很必要的。

研究电磁兼容技术的国际组织很多，如国际大电网会议（CIGRE）、国际供电会议（CIRDE）、电气电子工程师学会（IEEE）、国际电工委员会（IEC）、国际发供电联盟（UNIDEDE）、国际电报电话咨询委员会（CCITT）、国际无线电通信咨询委员会（CCIR）、国际通信联盟（ITU）、国际电热联盟（UIE）等。

其中 IEC 有两个专门从事电磁兼容标准化工作的技术委员会：国际无线电干扰特别委员会（CISPR，成立于 1934 年），下设 7 个分会；国际电磁兼容委员会（TC77，成立于 1981 年），下设 3 个分会。另外，在 IEC 中还有几十个产品委员会关注电磁兼容问题，例如 TC65“工业过程测量和控制技术委员会”制定了 IEC801《工业过程测量和控制装置的电磁兼容性》国际标准（已被 IEC1000-4 代替）。截止到 1997 年，IECTC77 已发布 27 个标准，其中 23 个是 IEC1000 系列标准。CISPR 最初关心的主要是广播接收频段和无线电干扰问题，近年来又涉及信息技术、工程、科研仪器、医疗设备、高压架空线路和高压设备之间电磁兼容技术，如车辆、家电、电动工具、电器产品等，工作频率多在 9kHz～18GHz，相应也开展了一些抗扰度标准研究。TC77 最初关心低压电网系统的 EMC（electromagnetic compatibility）标准（9kHz 以下），近两年扩展到整个 EMC 所涉及的产品。国际上的这两个专业委员会随技术发展和世界贸易需求已有一定交叉，IEC 正在从组织上和技术上采取积极措施加以协调解决，对此我国也给予高度关注。

TC77 的主要任务是为 IEC 的电磁兼容专家及产品委员会制备基本文件，即 IEC1000 系列标准，涉及到电磁环境、发射、抗扰性、试验程序和测量技术等规范。特别是处理与电力网络、控制网络以及其关联设备的 EMC 问题。TC77 负责所有频率范围内的抗扰性基础标准和在低频范围内（一般低于 9kHz）的发射干扰标准，其中也涉及通用标准和一些产品系列或产品标准。这些标准给出了在电磁环境中的发射者和敏感者之间的一些兼容性参数值，如发射电平、发射限值、抗扰电平、抗扰限值、兼容电平、抗扰裕量、发射裕量、电磁兼容裕度等，并提供了考核兼容程度的测量和试验方法。TC77 除上述任务外还与其他技术委员会协作制定其他一些标准，如 TC77、TC8 组成联合工作组，根据市场要求和现有标准情况提出具体的电力质量新工作项目提案、制定电力质量领域标准；与 TC12、TC85、CISPR 组成联合工作组，制订“高频电磁场对人体辐射的测量技术和程序”的标准等。

由 TC77 负责制定的 IEC1000 系列标准是近年来 IEC 出版的内容最为丰富的一个系列出版物，截至 1997 年 2 月已出版的国际标准、技术报告共计 30 篇，正在讨论或将要出版的共有 23 篇，还有若干篇尚在准备中。EMC IEC1000 系列包括总论、环境、限值试验和测量技术、安装和调试、通用标准、电能质量等，其中 IEC100024 是目前国际上比较完整的和系统的抗扰性基础标准，对其他 IEC EMC 标准的制订和修订有着重大影响，其试验方法、试验等级和测量技术形成了评估电工、电子产品的抗干扰能力的产品质量认证依据。IEC100025 主要考虑一些装置或设施的安装环境、安装方法、调试和减缓原则，以及装置的技术要求等，已出版第 1 篇、第 4 篇和第 5 篇。为了保证在工商业和居民区中使用的电气、电子设备及系统的电磁兼容，IEC10002521 提出关于减缓骚扰方法的原则，那些敏感的电气、电子设备和系统及具有高的骚扰发射电平的设备安装者以及制造商可以

在新的项目和现有设备的扩充和改造中采用这个原则。它包括了设备的抗扰性、设备端口处理、设备保护概念、减缓措施的必要性评估、对安装质量的评估和 EMC 检查等内容。

IEC1000 系列标准是基本 EMC 标准,它不但对装置、设施和系统的电磁辐射提出了要求,而且规定了考核其抗扰性的基本测量和试验方法。1997 年 2 月 14 日 IEC 正式成立了合格评定局(CAB)。CAB 限定成员国 12 个,中国 CSBTS 派代表参加工作。IEC 副主席作为 CAB 主席。在这次会议上正式讨论了"是否在 IEC 内设立 EMC 认证制度"的问题。由国际电工安全认证委员会对国际上的 24 个认证试验室能否承担 EMC 检测进行调查。调查结果表明,绝大部分原能承担产品安全检测的试验室也能承担 EMC 试验。经讨论 IEC 决定同意由 IECEE 先提出具体的 EMC 认证方案,这说明 IEC 确实要建立 EMC 国际认证制度。

我国是 IEC TC77 的成员国。1994 年为全面规划和推进我国 EMC 标准的制订、修订工作,促进国家电磁兼容技术进步和保护电磁环境,在全国无线电干扰委员会和国内 TC77 归口工作基础上,及时成立了《全国电磁兼容标准化联合工作组》,全面规划了以 IEC1000 系列为主的电磁兼容标准体系和具体目标,力争在两年多时间里完成这些标准的制定任务。我国正在制订、修订的 EMC 国家标准共有 23 项。在国家技术监督局的统一布置下,IEC1000 系列标准的转化工作已经开始,1999 年已经制订出一批等同或等效采用 IEC1000 的国家标准,并且绝大部分为强制执行标准,以解决电气、电子设备制造部门和用户的急需,同时也为规范我国 EMC 认证体系提供了依据。

12.3 微控制器系统的电磁兼容问题

在微控制器系统中,由于微控制器的时钟频率越来越高,电磁兼容的问题也越来越突出。印刷线路板上信号的高频成分可以通过元器件的引脚以及从印刷线路板到被驱动部件的引线发射出去,如同无线电天线将高频信号以空间电波的形式发送出去成为干扰其他电器设备的噪声源。信号在印刷线路板上通过元件间的连线传送,与该线平行且离该线很近的另一条引线上会产生感应噪声,引起对相邻线路的干扰,即一块印刷线路板上的信号间交叉干扰。如果微控制器系统中含有信号微弱的传感器或对精度要求较高的模拟电路,抗外界干扰以及抗来自系统内部的电磁干扰就尤其重要。

微控制器系统的抗干扰能力与印刷线路板的设计与布线有很大关系。理想的数字电路信号应该高低电平稳定,上升、下降沿整齐平稳,没有过冲与不到位,即为理想的方波。而当用示波器观察印刷线路板上的信号时,会发现有噪声信号叠加在所谓理想信号上。对接近理想波形的信号,俗称该信号比较"干净"。如果印刷线路板上的信号"不干净",即噪声较大,当外界来的随机电磁干扰信号叠加在原有的噪声上,就会造成系统出错。

虽然微控制器本身有一些抗干扰方面的措施与考虑,例如看门狗电路,在系统出错时复位,但是一个系统老出现看门狗电路复位动作,只能说明系统极不可靠。出错复位电路只能用于抵御外来的不可抗拒的巨大干扰,例如强雷击天气时,系统瞬时掉电等。

对于不得干扰其他电器设备的要求,各国对电器产品对环境造成的电磁干扰都有相应的规定。一个产品,要想进入国际市场,必须满足电磁兼容性测试要求,必须通过相应

的电磁兼容性检测。对不同类产品,电磁兼容性要求的指标是不一样的。即使是同类产品,对电磁兼容性的满足程度也有不同的级别。例如美国 FCC 规范中,B 级要比 A 级严格 10dB。

电磁兼容性的检测是在一个屏蔽很好的实验厅内,用专门的设备接收被测试产品发出的所有频率的电磁波信号。要求在任何一个频率上都不大于规定的值。

满足电磁兼容性要求,不是等到产品定型后才拿去检测,检测不过关时再寻找补救措施,而是要在整个产品开发期间都注意电磁干扰问题。特别是在印刷线路板设计元器件焊接工艺等方面予以足够的重视。

只有每个产品都满足相应的电磁兼容性测试标准,这些产品互相离得很近时,才不会引起互相干扰。

在下述 3 种情况下要特别注意抗电磁干扰问题。

(1) 微控制器时钟频率特别高、总线周期特别快的系统。

(2) 系统中含有大功率、大电流驱动电路的,如产生火花的继电器、大电流开关等。

(3) 含有微弱模拟信号电路以及高精度 A/D 变换电路的系统。

对于速度不高、模拟与数字电路共存问题不严重、功率器件有限的产品是不是就没有抗干扰或电磁兼容性方面的问题了呢? 这个问题要辩证地看,不注意抗电磁干扰问题,印刷线路板设计制造工艺太差,系统可靠性差显然是不行的。电磁兼容性问题,在采取了一些措施以后总是可以解决的。例如,4 层印刷线路板比双面板的噪声要低 10 倍,或者说低 20dB。4 层板的成本则比双面板高 2 倍。而由微控制器构成的一般系统,双面板的设计与制造中,只要充分考虑了抗电磁干扰与电磁兼容性问题,就完全可以满足应用方面的要求。而一个产品,特别是家电类产品,为降低成本,印刷线路板以单面板为主。在满足电磁兼容性要求的前提下,要节省每一个电容、电阻以降低成本。一个微控制器应用系统设计工程师面临的挑战是既要系统快、精度高、可靠、不干扰别的电器设备,还要成本最低。

12.4 噪声的来源与传输

本书中时序图上的信号波形是理想情况下的波形。当用示波器观察这些信号时,看到的实际上是信号与噪声的叠加。噪声是对信号的干扰,任何电子学系统都有噪声。信号对于电路的某一部分是有用的信号,它感应到电路的其他部分就成了噪声。

归纳起来,噪声来源于以下 3 个方面。

(1) 人为因素,如数字电路的开关过程、高频无线电信号的发射、继电器和开关的动作、马达的旋转、汽车火花的放电等。

(2) 大自然的噪声,如雷电、太阳黑子爆发等。

(3) 电子学器件的固有噪声、热噪声。

电子器件非线性引起的信号畸变不属于噪声范畴。

噪声是无法限制和去除的,而噪声引起的电磁干扰和对微控制器系统的影响则可以通过合理的印刷线路板设计和制造工艺减至最小,使之不构成对系统正常工作的威胁,且

满足电磁兼容性要求。

在印刷线路板设计与布线中要特别予以注意的是信号间的交叉干扰。时钟频率越高的微控制器,噪声问题越严重,这个说法并不确切。考虑同样频率的方波与正弦波,方波中的高频成分比正弦波中多得多。对方波作傅立叶变换得到与方波相应的正弦波基本频率的波和其整数倍频率波的叠加。虽然高频成分波的幅度比基波小,但频率越高,越容易发射出去成为噪声源。微控制器产生的最有影响的高频噪声的频率大约是其时钟频率的3倍。严格地讲,噪声的产生主要与时钟信号的上升时间和下降时间有关,即门电路的沿的跳变时间 Tr。

下面分析一个上升时间为 Tr 的信号通过一段长导线的情形。设导线引起的信号延迟为 Tpd,当 $Tpd \gg Tr$ 时,就成了一个传输线问题。如果导线另一端悬空,由于信号到达后能量无处释放,信号被发射回来,反射系数等于1。如果远端接地,反射系数等于-1。如远端接与导线阻抗相匹配的负载电阻,则无反射信号,信号无畸变地传到目的地。反射回来的信号就是传输过程中形成的噪声。

当 $Tr \gg Tpd$ 时,反射回来的信号叠加在信号的上升过程中,可以看作被输出端提供的功率吸收掉了,不必看成传输线问题。

当今的微控制器主要采用高速 CMOS 技术制造。信号输入端静态输入电流在 1μA 左右,输入电容在 10 pF 左右,输入阻抗相当高。高速 CMOS 电路的输出端都有相当的带负载能力,即相当大的扇出值。如果将一个门的输出端通过一段很长的引线连到输入阻抗相当高的输入端,反射问题就很严重。

当信号上升时间 Tr 小于线路的延迟时间 Tpd 时,要按照快电子学的要求处理。这是印刷板布线的一条原则。

信号在印刷线路板上的延迟时间与引线的特性阻抗有关,即与印刷线路板材料的介电常数有关。可以粗略地认为,信号在印刷板引线上的传播速度约为光速的1/3~1/2。

表 12-1 各类逻辑电路上升时间

逻辑电路	上升时间 Tr/ns	逻辑电路	上升时间 Tr/ns
GaAS	0.1	74ALS	6
ECL	0.75	74LS	8
74F	3	74HCT	8
74AS	3	74HC	18
74AC	4		

表12-1列出了常用逻辑电路的标称上升时间 Tr。可见,微控制器系统中,常用逻辑电路元件的 Tr 在4~20ns之间。

在印刷线路板上,信号通过一个7Ω的电阻和一段25cm长的引线,线上的延迟时间 Tpd 大致就在4~20 ns之间。也就是说,信号在印刷线路板上的引线长度越短越好,最长也不宜超过25cm。

12.5 信号线间的交叉干扰

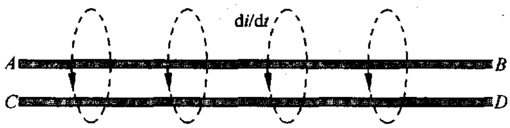


图 12-1 信号线间交叉干扰的示意

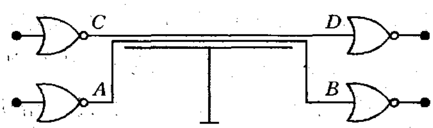


图 12-2 信号线间交叉干扰

在图 12-1 中，导线 AB 中有随时间变化的电流（di/dt）流过，于是 AB 线的周围产生了变化的磁场。导线 CD 在变化的磁场中，于是形成了 AB 线对 CD 线的干扰。

图 12-2 中 AB 和 CD 是两根平行的相距很近的信号线。图 12-4 是图 12-2 的电原理等效电路。图 12-3 示意信号延迟时间远大于信号上升时间时，A、B、C、D 点的波形情况。A 点有一个上升时间为 $T\mathrm{r}$ 的阶跃信号通过引线 AB 传向 B 端。信号在 AB 线上的延迟时间是 $T\mathrm{d}$。由于 A 点信号的向前传输、到达 B 点后的信号反射和 AB 线的延迟，$T\mathrm{d}$ 时间以后在 D 点会感应出一个宽度为 $T\mathrm{r}$ 的负脉冲信号，在 C 点，由于 AB 上信号的传输与反射，会感应出一个 $2T\mathrm{d}$ 的正脉冲信号。这就是信号间的交叉干扰。干扰信号的强度与 A 点信号的 di/dt 有关，与线间距离有关。当两信号线不是很长时，CD 上看到的实际是两个脉冲的叠加。

CMOS 工艺制造的微控制器输入阻抗高，噪声高，噪声容限也高，数字电路上叠加 100～200mV 噪声并不影响其工作。若图 12-2 中 CD 线是一模拟信号。这个干扰就会变得不可容忍。

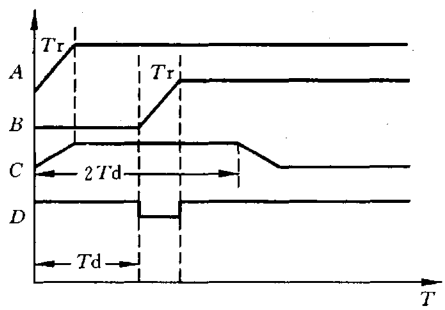


图 12-3 $T\mathrm{d} \gg T\mathrm{r}$ 时信号线间交叉干扰的波形分析

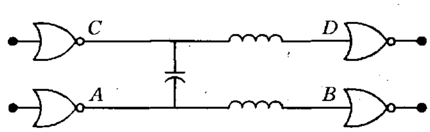


图 12-4 信号线间交叉干扰等效电路干扰

当印刷线路板是四层板，其中有一层是大面积的地，或当印刷线路板是双面板，信号线的反面是大面积的地时，这种信号间的交叉干扰就会变小。原因是，大面积的地减小了信号线的特性阻抗。信号在 B 端的反射大大减小。特性阻抗与信号线到地间介质的介电常数的平方成反比，与介质厚度的自然对数成正比。

若 CD 线为一模拟信号，要避免数字电路信号线 AB 对 CD 的干扰，AB、CD 线下方要有大面积的地，AB 线到 CD 线的距离要比 AB、CD 线到地的距离远 2～3 倍。

12.6 来自电源的噪声

电源是向系统提供能量的。电源在提供能源的同时，会直接由供电端将噪声加在微控制器上。微控制器的复位线、中断线和其他一些控制线最容易受到外界噪声的干扰。电网上的强干扰通过电源进入集成电路块，即使是电池供电的系统，电池本身也有高频噪声。模拟电路中的模拟信号放大器则更经不起来自电源的干扰。

图 12-5 是噪声通过电源与地进入集成电路的示意图。集成电路块 A 与 B 的电源线、地线都有公共部分。设 A 是一片常用的 8 路门电路，若每个门在打开、关闭时的电流变化为 4mA，则 8 个门同时打开、关闭时的电流变化就为 32mA。若信号上升时间是 5ns，接地线的等效电感是 50nH，则噪声 V 可用以下公式计算：

$$V = L\frac{\mathrm{d}i}{\mathrm{d}t} = 50\mathrm{nH}\,\frac{32\mathrm{mA}}{5\mathrm{ns}} = 0.32\mathrm{V}$$

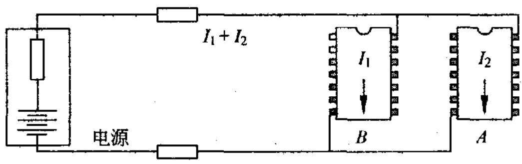


图 12-5　印刷板布线不合理引起的噪声

也就是说，由于 A 与 B 有一段公共地线，A 在开关过程中产生 5ns 宽、峰值 0.32V 的噪声直接进到 B 中，对集成电路 B 产生干扰。

以上分析也适用电源线上的一段公共阻抗部分。为了减小噪声通过电源线、地线的干扰，在印刷线路板布线时，要尽量减小公共部分的阻抗。然而电源的内阻是去不掉的，干扰仍会通过电源引起。

图 12-5 中，集成电路 A 与电源形成了一个电流的回路。这个电流回路会产生磁场。电流又在不断高速变化着，回路以内就产生了复杂变化的电场、磁场。集成电路 B 恰恰在这个回路形成的环中，B 受到 A 的干扰。如果把集成电路 B 摆放在电源与 A 形成的电流环路外面，A 对 B 的干扰程度就会大大降低。

12.7 印刷线路板与元器件的高频特性

在微控制器应用系统中，元器件的上升下降时间越快，引入系统的噪声问题就越严重，噪声的高频成分就越多。这是因为上升时间越快，电流随时间的变化率 $\mathrm{d}i/\mathrm{d}t$ 就越大。对微控制器系统影响最大的噪声频率一般是系统时钟频率的 3 倍。如果微控制器使用了 5MHz 的石英振荡器或陶瓷振荡器，最可能发射出的噪声频率大约是 15MHz。

在高频情况下，印刷线路板上的引线、过孔、电阻、电容、接插件的分布电感与电容变得不可忽视；而且，电容的分布电感，电感的分布电容都不可忽视；电阻产生对高频信号的反射，引线的分布电感会起作用，尤其是当引线长度大于噪声频率相应波长的 1/20 时，引

线能像天线一样将噪声发射出去，这就是所谓天线效应。

一个印刷线路板上的过孔大约引起 0.6pF 的电容。

一个集成电路本身的封装材料引入 2～10pF 的分布电容。

一个线路板上的接插件有 5～20nH 的分布电感。

一个双列直插的 24 引脚集成电路插座，引入 4～18nH 的分布电感。

这些小的分布参数对于运行在较低频率下的微控制器系统是可以忽略不计的，而对于高速系统，必须给予特别的注意。

什么叫高速系统呢？当信号的上升时间接近信号的延迟时间时，就构成了高速系统，就要按照高速信号来处理。对高速信号的传输，要考虑传输线的阻抗匹配问题。对于一块印刷线路板上的集成块之间的信号传输，印刷板上的引线一般是不作传输线考虑的，所以印刷线路板大时，系统的速度就不能太快。

如图 12-6 所示，在一块印制板上，假设信号线的传输距离是 25cm（直线或折线），信号源内阻为 7Ω 时，信号在线上的延迟时间大约等于 4～20ns。

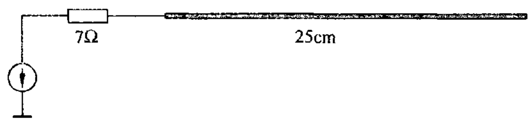


图 12-6 信号通过 25cm 悬空线

在高速微控制器系统中，逻辑电路的门的延迟时间大约在 4～20ns 之间，大致相当于一段 25cm 长的引线的延迟时间。印刷板上的信号的最大传输路程，应控制在 25cm 以内，而且其上的过孔数目应尽量少，最好不多于 2 个。

设计印刷线路板的一个规则是：信号在印刷板上传输，线的延迟时间不应大于所用器件的标称上升时间。

12.8 元件的布置

在印刷线路板上排列元件时，要充分考虑抗电磁干扰问题。原则之一是各部件之间的引线要尽量短。在布局上，要把模拟信号部分、高速数字电路部分、噪声源部分（继电器、大电流开关等）这三部分合理地分开，使相互间的信号耦合减到最小，如图 12-7 所示。

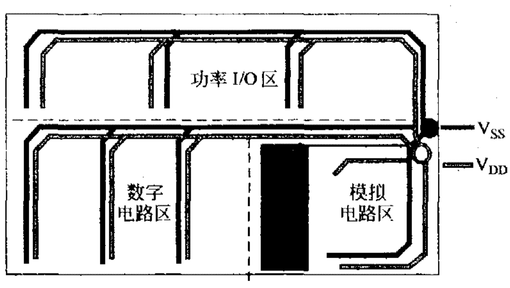


图 12-7 印刷线路板上的合理分区

12.9 印刷线路板接地线的处理

每个集成电路电源、地形成的回路圈要尽量小。

印刷线路板上的线，以电源线和地线最重要。克服电磁干扰问题，最主要的手段就是合理接地。

微控制器系统要有良好的抗干扰能力和满足电磁兼容的要求，最好使用多层印刷线路板，特别是对时钟信号在 10 MHz 以上的微控制器。四层板效果要好得多。

四层印刷线路板是在常用的双面板的基础上，在元件面走线层和焊接面走线层之间增了电源层和地层，使双面板上原来只能用粗线布线的电源与地线变成一个大面积的地层和电源层。

举例来说，假设双面板上有一条地线，它的长×宽×厚＝100mm×1mm×0.03mm，当频率为 7MHz 时，这段线路的阻抗大约是 300mΩ，而大平面的地线阻抗只有 1 mΩ，对于 300mV 的噪声，在大平面地的情况下就减到 1mV。

一般来说，四层板要比双面板的噪声水平低 20dB，六层板又比四层板低 10dB。

但是这并不是说使用了多层板以后，抗电磁干扰问题上就没有什么要注意的了。在多层板情况下，对于吞吐电流特别大的器件，它的电源端和地端往往不直接接电源层和地层，而是在该元件的附近放一个充放电电容。电容的引脚接电源层和地层，而该元件的电源、地则用粗线与电容相连。

对于双面板，地线要布置得特别考究，以抑制电磁干扰。在印刷板布线上，通常采用单点接地法。电源和地是从电源的两端接到印刷线路板上来的。电源一个接点，地一个接点。在印刷线路板布线上要有多条返回地线，这些线都汇聚到电源的那个接地点上，就是所谓单点接地。所谓模拟地线、数字地线、大功率器件地线分开，指布线时地线分开，而最后都汇集到这个接地点上。

为扼制大功率器件对微控制器部分数字电路的干扰以及数字电路对模拟电路的干扰，两类不同的地在接向公共接地点时要用高频扼流环，俗称磁珠。这种器件对低频信号可以看成电阻为零，对高频干扰信号可以看成是一个电感。

注意，不能使用电感来作接地用的高频扼流器件。因为电感器件通常有较大的直流电阻。

我们用 SPICE 仿真软件模拟过门电路开关时在印刷线路板的地线上引起的噪声。考虑一个 64 位的数据总线，在 64 个门同时开关时，远离接地点的那片集成电路的地线上引起的噪声高达 1.5V。如果这 64 个门共用一条返回地线的话，远端的片子将无法工作。解决办法是采用多条接地线，返回到一个点上。

与印刷线路板以外的信号线相连时，通常采用屏蔽电缆。对于高频信号和数字信号，屏蔽电缆的两端都接地。低频模拟信号用的屏蔽电缆，一端接地为好。

若微控制器系统最终要驱动大电流的设备或信号高频成分含量很高，例如驱动 CRT 显示器，从印刷线路板到阴极射线管要有一条电缆，这条电缆会像天线一样将高频干扰发射出去。这条电缆必须屏蔽得很好，不能有露出来的地方，屏蔽要接到地线上，否则就达

不到电磁兼容性要求的指标。

对噪声和干扰非常敏感的电路或高频噪声特别严重的电路，应该用金属罩屏蔽。铁磁屏蔽对 500kHz 的高频噪声效果并不明显，薄铜皮屏蔽效果要好一些。使用螺丝钉固定屏蔽罩时，要注意不同材料接触时引起的电位差造成的腐蚀。

对于印刷线路板上的敏感模拟信号的输入端引线，可用局部加屏蔽地的办法予以保护。办法是在引线的反面加屏蔽地。在有引线的一面，引线左右两侧也要布以地线，并且让该信号线与离得最近的其他信号线之间的距离比该信号线到地的距离大 2 至 3 倍，就可以防止信号的交叉干扰。

12.10 抑制噪声的器件

首先是去耦电容。微控制器时钟电路采用方波而不是正弦波。方波比正弦波含有更高的高频成分。对电磁干扰贡献最大的噪声频率是相当于时钟频率 3 倍的高频噪声。噪声中也还有频率更高的成分。在频率很高时，对电容来说，它的寄生电感不能忽视；对电感来说，它的分布电容会起作用；对电阻来说，它存在着对高频信号的反射；对印刷板上的引线来说，它的分布电感不容忽视，而且长的引线还表现出天线的特征，将高频噪声信号发射出去。焊接印刷线路板上的元件时，如果电阻、电容、三极管的引脚留得很长，这些长的引脚恰好就成了高频信号的天线。特别是印刷线路板上的信号引出线，如果该信号引出线的长度达到高频噪声信号波长的 1/20，噪声信号就能被发射出去。例如，对于 100MHz 的噪声信号，波长是 3m，15cm 的引线就是它的发射天线。门电路中，各门频繁地导通、截止，引起集成电路电源、地线上电流变化率 di/dt 相当大，也会产生相当大的高频电磁干扰。去除这些高频噪声和干扰的办法是加去耦电容或高频旁路电容。

好的高频去耦电容可以去除高至 1GHz 的高频成分。陶瓷片电容或多层陶瓷电容高频特性比较好，如常用的 25U 系列。设计印刷线路板时，每个集成电路的电源、地之间都要加一个去耦电容。

这个去耦电容有两个作用，一方面是充当本集成电路的蓄能电容，提供和吸收该集成电路开门、关门瞬间的充、放电能量；另一方面是旁路掉该器件的高频噪声。数字电路中典型的去耦电容值是 0.1μF。而这种电容由于本身结构的特点以及有一段引脚，免不了有分布电感。这个分布电感的典型值是 5nH。即使是表面封装的去耦电容也免不了有分布电感。有 5nH 分布电感的去耦电容，并行共振频率大约在 7MHz 左右。也就是说，它对于 10MHz 以下的噪声有较好的去耦作用；对 40MHz 以上的噪声几乎不起作用。

1nF、10nF 的电容，并行共振频率在 20MHz 以上，去除高频噪声的效果要好一些。在电源一进入印制板的地方加一个 1nF 或 10nF 的去高频电容往往是有利的。即使是用电池供电的系统，也需要这种电容。

每 10 片左右集成电路要加一片充放电电容，或称蓄能电容，电容值可选 10μF 左右。最好使用钽电容或聚碳酸酯电容，不要用电解电容，因为电解电容是用两层薄膜卷起来的，这种卷起来的结构在高频时表现为电感。

去耦电容值的选用并不严格，可按 $C\times F=1$ 选用，即 10MHz 取 0.1μF；100MHz 取

0.01μF；对由微控制器构成的系统，取 0.1～0.01μF 之间都可以。

特别要注意的是，在焊接时去耦电容的引脚要尽量短。长的引脚会使去耦电容本身发生自共振。表 12-2 给出了瓷片电容的自共振频率与引脚长度的关系。

表 12-2 瓷片电容的自共振频率与引脚长度的关系

引脚长度	1/4 英寸	1/2 英寸
10 000pF	12MHz	11MHz
1000pF	35MHz	32MHz
100pF	150MHz	120MHz
10pF	500MHz	350MHz

常用的抑制噪声的器件还有磁珠。磁珠由高频铁氧体材料制成，多为圆柱形，轴向有 3、4 个通孔，使用时用尽量粗的铜线穿入通孔，并来回穿上 1 至 2 圈，串连在电源线地线上，用以扼制线上的高频干扰。由于铁氧体材料的磁阻，既导磁又有损耗，它可以将高频噪声的能量在铁氧体内部转换成热能，即以热能的形式将高频噪声吸收掉。磁珠常常用来隔离印刷线路板上的几个分区，如模拟电路区、数字电路区、功率器件区等。

为微控制器系统的电源配制适当的低通滤波器也是防止来自电源线干扰的有效手段。低通滤波器的主要参数是插入损耗，使用的低通滤波器如果插入损耗过小，可能达不到扼制噪声的要求，而过高的插入损耗可能会引起漏电，在人身安全方面不符合要求。故使用低通滤波器时要注意参数的选取。

近年来，还出现了一些新的 EMC 器件，如 TVS、TVP 等。TVS 是英文字 transient voltage supervision 的缩写，意为瞬变电压抑制器。TVS 器件可以理解为一只高速的齐纳二极管，或是两只对顶的、中心极接地的齐纳二极管，接在电源线与地线之间，它平时不导通，当电源线上出现瞬间强干扰时，TVS 器件快速导通，将强干扰信号短路掉。

如果微控制器系统是用于室外的，还要考虑防雷击问题，防雷击器件有压敏电阻、自恢复保险丝等。

12.11 控制噪声的经验

12.11.1 控制噪声源

控制噪声源可采用如下方法：

(1) 能用低速芯片就不用高速的，高速芯片只用在关键地方。

(2) 一片 74HC04 中有 6 个非门，如果时钟电路用了其中的 2 个，另外 4 个尽量用在不重要的地方，尤其不要用在 I/O 驱动上。

(3) 可用串一个电阻的办法，降低控制电路沿上下跳变速率。

(4) 尽量为继电器提供某种形式的阻尼。

(5) 使用满足系统要求的最低频率时钟。

(6) 时钟产生器尽量靠近用到该时钟的器件。

(7) 石英晶体振荡器外壳要接地。

(8) 尽量让时钟信号回路周围电场趋近于零。用地线将时钟区圈起来,时钟线要尽量短,不要引得到处都是。

(9) I/O 驱动电路尽量靠近印制板边,让它尽快离开印制板。

(10) 对进入印制板的信号要加滤波。

(11) 从高噪声区来的信号也要加滤波。

(12) 闲置不用的门电路输入端不要悬空,闲置不用的运算放大器正输入端要接地,负输入端接输出端。

(13) 使用 45°折线而不要用 90°折线布线,以减小高频信号对外的发射。

12.11.2 减小噪声的耦合

减小噪声的耦合可采用如下方法:

(1) 在印刷线路板上按频率和电流开关特性分区,噪声元件与非噪声元件要离得远一些。

(2) 对特殊高速逻辑电路部分用地线圈起来。

(3) I/O 片子靠近印刷线路板边,靠近引出插头。

(4) 经济条件允许的话用多层板,以减小电源、地的寄生电感。

(5) 单面板或双面板用单点接电源和单点接地。

(6) 电源线、地线尽量粗。

(7) 时钟、总线、片选信号要远离 I/O 线和接插件。

(8) 模拟电压输入线、参考电压端要尽量远离数字电路信号线,特别是时钟线。

(9) 对 A/D 类器件,数字部分与模拟部分宁可绕一下也不要交叉。

(10) 时钟线垂直于 I/O 线比平行于 I/O 线干扰小,时钟元件引脚远离 I/O 电缆。

(11) 元件引脚要尽量短,去耦电容引脚要尽量短。

(12) 关键的线要尽量粗,并在两边加上保护地。

(13) 噪声敏感线不要与大电流、高速开关线平行。

(14) 高速线要短要直。

(15) 石英晶振下面和对噪声特别敏感的器件下面不要走线。

(16) 敏感信号与噪声携带信号要通过一个接插件引出的话,例如用扁带电缆引出,要使用地线—信号线—地线的引出法。

(17) 弱信号电路、低频电路周围地线不要形成环路。

(18) 携带高噪声的引出线要绞起来,最好屏蔽起来。

(19) 集成电路上该接电源、地的端都要接,不要悬空。

12.11.3 减小噪声接收

减小噪声接收可采用如下方法：

(1) 任何信号都不要形成环路，如不可避免，让环路区尽量小。

(2) 使用高频、低寄生电感的瓷片电容或多层陶瓷电容作去耦电容。

(3) 每个集成电路加一个去耦电容。

(4) 用大容量的钽电容或聚酯电容而不用电解电容作电路充放电储能电容。

(5) 每个电解电容边上都要加一个小的高频旁路电容。

(6) 需要时，线路中加铁氧体高频扼流环分离信号、噪声、电源、地。

(7) 可能的话，加频率可选的带通滤波器。

(8) 使用管状电容时，外壳要接地。

(9) 微控制器无用端要接高电平、接地，或定义成输出端。

(10) A/D 参考电平要加去耦电容，用串联终端电阻的方法减小信号传输中的反射。

(11) 尽量不用 IC 插座，而是将集成电路，特别是高性能的模拟电路器件和数字、模拟混合的集成电路直接焊在印刷线路板上。

第13章　嵌入式实时系统在MC68HC08上的实现

13.1　什么是嵌入式实时操作系统

提到计算机的操作系统，人们马上会想到DOS、Windows98、Unix等等。用户通过操作系统同计算机打交道。没有操作系统，计算机就无法使用。操作系统的主要任务可归结为以下4个方面。

(1) 任务管理，逐个管理计算机中的每个进程或任务。

(2) 内存管理，管理计算机内部存储空间的分配。

(3) I/O管理，管理与计算机相关的输入输出设备。

(4) 文件管理，管理外部存储媒体中的文件系统。

PC机的操作系统中，DOS是典型的单用户单任务系统，Windows98是单用户、多任务系统，而Unix、Linux是多用户、多任务系统。上述这些系统，都不是实时系统。

如果多任务系统中任务太多，或多用户系统中用计算机的用户过多，计算机就会变得很慢。在嵌入式系统中，计算机是面向控制的。对于控制系统来说，实时性最重要，而有没有操作系统并不重要。实时的含义是，计算机对外部事件的响应时间必须确保小于某一确定的值。可以认为实时性是嵌入式系统最重要的指标。

嵌入式应用中，可以不使用任何操作系统，即下面提到的前后台系统。即使使用嵌入式实时操作系统，操作系统的概念也和上述意义上的操作系统有很多不同。嵌入式实时操作系统实际上是管理应用程序定义好的任务，而不是面向用户或者说面向人的，故相对来说要简单一些。嵌入式应用中一般没有外部存储媒体，故一般没有文件系统。小的应用系统也没有内存管理。在I/O管理上，嵌入式实时操作系统是把I/O当作若干个分离的任务来管理的，因此嵌入式实时操作系统的任务主要是多任务的管理，而多任务的管理是通过在嵌入式系统中运行一个实时内核实现的。故实时内核是嵌入式实时操作系统的核心，它对多个任务进行调度、管理、保证应用程序的实时性。

13.2　3种嵌入式实时系统

嵌入式实时系统可分为3类：前后台系统、使用不可剥夺型内核的多任务实时系统和使用可剥夺型内核的多任务实时系统。

13.2.1 前后台系统

前后台系统是指不使用实时操作系统的应用程序。一般做法是，写一个无限循环的程序，在程序中查询每个事件是否发生，每个任务是否具备运行条件，如果是，则处理这个事件或执行这个任务。这可以看成是一个后台程序。

而系统对中断的响应可以看成是前台程序，中断的打入表示某事件的发生，在中断服务子程序中，一般对中断做一些简单的处理，然后给出一个某事件发生了的标志，待后台程序来处理。后台可以叫作任务级，前台叫作中断级。这种系统在处理信息的及时性上，比实际可能做到的要差。这个从事件发生到开始处理的时间指标称为系统的任务级响应。最坏情况下的系统任务级响应时间取决于整个循环的执行时间。因为循环的执行时间不是常数，程序经过某一特定部分的准确时间也是不能确定的。

13.2.2 使用不可剥夺型内核的多任务实时系统

不可剥夺型内核是指每个任务应自我放弃 CPU 的所有权，不可剥夺型调度法也称作合作型多任务，各个任务彼此合作共享一个 CPU。异步事件还是由中断服务来处理。中断服务可以使一个高优先级的任务由挂起状态变为就绪状态。但中断服务以后控制权还是回到原来被中断了的那个任务，直到该任务主动放弃 CPU 的使用权时，那个高优先级的任务才能获得 CPU 的使用权。

不可剥夺型内核的一个优点是响应中断快。在任务级，不可剥夺型内核允许使用不可重入函数，函数的可重入性并不重要。每个任务都可以调用不可重入性函数，而不必担心其他任务可能正在使用该函数，从而造成数据的破坏。因为每个任务要运行到完成时才释放 CPU 的控制权。当然被调用的函数本身不得有放弃 CPU 控制权的企图。

使用不可剥夺型内核时，任务级响应时间比前后台系统快得多。此时的任务级响应时间取决于最长的任务执行时间。比起下面的剥夺型内核来，不可剥夺型内核最大的优点是简单，且占用内存，特别是 RAM，要少很多。

不可剥夺型内核的另一个优点是，几乎不需要使用信号量保护共享数据。运行着的任务拥有自己的 CPU，不必担心被别的任务抢走。但这也不是绝对的，在某种情况下，信号量还是用得着的。处理共享 I/O 设备时仍需要使用互斥型信号量，例如，在打印机的使用上，仍需要满足互斥条件。

不可剥夺型内核的最大缺陷在于其响应时间，高优先级的任务已经进入就绪态，但还不能运行，要等，也许要等很长时间，直到当前运行着的任务释放 CPU。与前后台系统一样，不可剥夺型内核的任务级响应时间是不确定的，不知道什么时候最高优先级的任务才能拿到 CPU 的控制权，完全取决于应用程序什么时候释放 CPU。

总之，不可剥夺型内核允许每个任务运行，直到该任务自愿放弃 CPU 的控制权。中断可以打入运行着的任务，中断服务完成以后将 CPU 控制权还给被中断了的任务。任务级响应时间要大大好于前后台系统，但仍是不可知的，商业软件几乎没有不可剥夺型

内核。

13.2.3 使用可剥夺型内核的多任务实时系统

当系统响应时间很重要时，要使用可剥夺型内核。最高优先级的任务一旦就绪，总能得到 CPU 的控制权。当一个运行着的任务使一个比它优先级高的任务进入了就绪态，当前任务的 CPU 使用权就被剥夺了，或者说任务被挂起了，那个高优先级的任务立刻得到了 CPU 的控制权。如果是中断服务子程序使一个高优先级的任务进入就绪态，中断完成时，中断了的任务被挂起，优先级高的那个任务开始运行。

使用可剥夺性内核，最高优先级的任务什么时候可以执行，可以得到 CPU 的控制权是可知的。使用可剥夺型内核使得任务级响应时间得以最优化。

使用可剥夺型内核时，应用程序不应直接使用不可重入性函数，调用不可重入性函数时，要满足互斥条件，这一点可以用互斥型信号量来实现。如果调用不可重入性函数时，低优先级的任务 CPU 的使用权被高优先级任务剥夺，不可重入性函数中的数据有可能被破坏。因此，使用的 C 编译器所提供的原函数应具备可重入性。综上所述，可剥夺型内核总是让就绪态的高优先级的任务先运行，中断服务程序可以抢占 CPU，到中断服务完成时，内核让此时优先级最高的任务运行(不一定是那个被中断了的任务)。任务级系统响应时间得到了优化，且是可知的。

13.3 使用嵌入式实时操作系统的优点和缺点

13.3.1 使用嵌入式实时操作系统的优点

在嵌入式应用中使用实时操作系统有许多优点，主要表现以下 4 个方面。

(1) 将应用程序分解为若干个任务，各任务可单独编程，这样就大大简化了应用系统的程序设计。

(2) 实时性好，特别是使用可剥夺型实时内核。

(3) 有助于提高系统的稳定性和可靠性。

(4) 系统的软件维护和功能扩展较前后台系统方便很多。

13.3.2 使用嵌入式实时操作系统的缺点

使用嵌入式实时操作系统的缺点主要有

(1) 实时内核本身要占用一定数量的额外内存，包括 ROM 和 RAM。

(2) CPU 要增加大约 2%～5%的额外负荷。

(3) 使用商业 RTOS 软件，可能增加额外的成本。

13.4 在 MC68HC08 CPU 上实现实时内核的运行

对于 MC68HC05 CPU，由于堆栈只有 64 个字节，且地址是固定的，所以不能使用高级语言，也不可能使用可剥夺型实时内核。虽然在 Motorola 网页上有两个不可剥夺型实时内核是为 MC68HC05 写的，但在大量应用项目中用得并不多。本书将这两个实时内核移植到 MC68HC08 CPU 上运行。

当 MC68HC08 CPU 推出以后，由于堆栈活了，增加了栈处理指令，高级语言用得上，可以使用不可剥夺型内核。理论上说也可以使用可剥夺型实时内核。

下面给出 3 个实时内核在 MC68HC08 CPU 上运行实例的源代码。前 2 个是不可剥夺型的。其中，一个是基于优先级的，另一个是基于时间片的。这 2 个内核的前身是 Motorola 公司的 Joanne Santangeli 为 MC68HC05 CPU 编写的 2 个内核，我们在它们的基础上针对 MC68HC08 CPU 的特性进行了修改和优化。最后一个是可剥夺型的。可剥夺型内核只给出了硬件相关部分的源代码，源代码的其他部分见《μC/OS the Real Time Kernel》一书及其中译本《μC/OS-Ⅱ——源码公开的实时嵌入式操作系统》。

13.5 在 MC68HC08 CPU 上运行一个基于优先级的不可剥夺型实时内核

本内核将所有任务的优先级分为 3 个档次，每一档又有 8 个优先级，所以本内核同时共支持 24 个优先级，也就是 24 个任务。在编程时，用户需要为每个任务都赋予相应的优先级，并将程序入口地址写在任务表内。每个任务入口地址在任务表中的位置与它的优先级是一一对应的。任务就绪时，内核就是根据它的优先级在任务表中查找到相应的入口地址并执行的。

内核的运行是由不断的查询－保存－运行的过程组成的。内核在运行过程中，不断地查询各个任务的状态，一旦有任务就绪，内核就会从就绪的任务中选择优先级最高的任务(暂时称之为任务 1)开始执行，同时保存与当前任务优先级同档次的 8 个任务的状态。当任务 1 执行完成后，内核会查询刚才保存的任务状态，如果还有较低优先级的任务(暂时称之为任务 2)就绪，内核会立刻执行任务 2，直到同优先级档次内的所有就绪任务都执行完毕。需要注意的是，如果在任务状态保存后有某个任务(我们暂时称之为任务 3)进入就绪状态，由于这一变化并不会影响预先保存的结果，所以任务 3 暂时不会被执行，它必须等到下一次同优先级档次的就绪任务被内核顺序执行时，才会被执行。

内核会不断重复上述过程。而在这个循环过程中，如果最高优先级档次的 8 个任务不断地有任务就绪，那么较低档次的 16 个任务便永远得不到运行的机会。所以用户在编程时需要慎重考虑每个任务的优先级设置，确保内核的任务调度是正确和有效的。一般用户需要将频繁发生的任务的优先级降低，使得每个任务都能得到运行的机会。

以下是一个在 MC68HC908GP32 上使用本内核的例子。这段代码是在 MC68HC908GP32 IDK 在线编程开发系统上调试通过的。如果用户需要将它移植到其

他的仿真器系统或是自己的目标系统，需要添加芯片的初始化代码和中断向量表。

```
*************************************
* 内存和端口地址定义 *
*************************************
ROM       EQU    $8000          ;ROM 地址
RAM       EQU    $B0            ;RAM 地址
VECTOR    EQU    $FFF4          ;向量表地址
TABLE     EQU    $8400          ;任务表地址
PORTA     EQU    $00            ;端口 A 数据寄存器
DDRA      EQU    $04            ;端口 A 方向寄存器
PORTB     EQU    $01            ;端口 B 数据寄存器
DDRB      EQU    $05            ;端口 B 方向寄存器
BRATE     EQU    $0019          ;波特率寄存器
SCCR1     EQU    $0013          ;SCI 控制寄存器 1
b_LOOPS   EQU    7
b_ENSCI   EQU    6
b_TXINV   EQU    5
b_M       EQU    4
b_WAKE    EQU    3
b_ILTY    EQU    2
b_PEN     EQU    1
b_PTY     EQU    0
SCCR2     EQU    $0014          ;SCI 控制寄存器 2
b_SCTIE   EQU    7
b_TCIE    EQU    6
b_SCRIE   EQU    5
b_ILIE    EQU    4
b_TE      EQU    3
b_RE      EQU    2
b_RWU     EQU    1
b_SBK     EQU    0
SCDAT     EQU    $0018          ;SCI 数据寄存器
SCSR      EQU    $0016          ;SCI 状态寄存器
b_TDRF    EQU    7
b_TC      EQU    6
b_RDRF    EQU    5
*
CONFIG1   EQU    $001e
CONFIG2   EQU    $001f
*
PCTL      EQU    $0036
```

```
b_PLLON     EQU     5
b_BCS       EQU     4
PBWC        EQU     $0037
b_AUTO      EQU     7
b_LOCK      EQU     6
PMSH        EQU     $0038
PMSL        EQU     $0039
PMRS        EQU     $003a
PMDS        EQU     $003b
*
**************************************
* 优先级调度使用的常量 *
**************************************
LSB         EQU     0                   ;任务请求寄存器的第0位
DO_TASK     EQU     1                   ;表示执行优先级1的任务的标志
TRY_PR3     EQU     2                   ;表示检查优先级3的标志
GO_PR1      EQU     3                   ;表示返回优先级1的标志
*****************************
* 样板任务使用的常量 *
*****************************
FINAL       EQU     4
      ORG     RAM
********************************
* 优先级调度使用的变量 *
********************************
JUMPLONG            RMB     8       ;在RAM中预留的可以写入一个简短函数的空间
PR_LEVEL            RMB     1       ;存储优先级数
TASKREQ             RMB     3       ;任务请求寄存器
SHADOWTASK          RMB     3       ;任务请求寄存器的镜像
ADD_POINTER         RMB     1       ;指向任务表中地址的指针
SHIFTCNT            RMB     3       ;
TASKTEMP            RMB     1       ;为BRSET准备的SHADOWTASK的拷贝
SYSFLAG             RMB     1       ;系统保持标志
SETTASKS            RMB     1       ;在SCI子程序中设置要运行的任务

**************************
* 样本任务的变量 *
**************************
DELAY_VAR           RMB     1       ;
TIME_ON             RMB     1       ;
```

```
NUM_ON_LEDS     RMB     1       ;控制发光二极管的顺序
APP_FLAG_REG    RMB     1       ;
TEMP            RMB     1       ;用于 SCI 中断服务例程
TEMPLO          RMB     1       ;用于 SCI 中断服务例程
TEMPHI          RMB     1       ;用于 SCI 中断服务例程
    ORG         ROM
****************
* 主程序 *
****************
SCHED05
    JSR     INITIAL             ;初始化端口 A 和 RAM
    CLI                         ;清除中断屏蔽位
SCHED99
    JMP     PSCHED              ;优先级调度
****************
* 子程序 *
****************
* 名称: INITIAL
* 功能:  初始化所有的端口和 RAM
****************************************************************************
INITIAL
    CLR     PORTA               ;将端口 A 清零
    LDA     #$FF                ;设置端口 A 为输出
    STA     DDRA                ;
    JSR     CLEAR               ;将 RAM 清零
    RTS
CLEAR
    LDHX    #$0000 ;
CLEAR08
    CLR     RAM,X               ;将 RAM 清零
    AIX     #1                  ;到下一个字节空间
    CPHX    #$20                ;是否所有字节空间都已清零
    BLO     CLEAR08             ;如果没有,继续将下一个字节空间清零
    RTS                         ;否则退出
****************************************************************************
* 名称: PSCHED
* 功能: 这个例程负责进行基于优先级的任务调度
**********************************************************************
PSCHED
```

```
    JSR     PRIOR_1                 ;检查并执行优先级为 1 的任务
PSCHED05
    JSR     PRIOR_2                 ;检查优先级为 2 的任务请求寄存器
PSCHED10
    JSR     PRIOR_2OR3              ;执行某一个优先级为 2 或 3 的任务
    BRSET   TRY_PR3,SYSFLAG,PSCHED15  ;检查优先级为 3 的任务
    BRA     PSCHED                  ;返回到对优先级为 1 的任务的检查
PSCHED15
    JSR     PRIOR_3                 ;检查优先级为 3 的任务
PSCHED99
    BRA     PSCHED10                ;执行某一个优先级为 2 或 3 的任务
*******************************************************************
* 名称：PRIOR_ 1
* 功能：检查优先级为 1 的任务请求寄存器，并逐一执行所有准备就绪的任务
*******************************************************************
PRIOR_1
    CLRX    ;
    STX     PR_LEVEL                ;设定优先级为 1
    JSR     COPY                    ;复制任务请求寄存器到一个临时内存空间
    LDA     SHADOWTASK,X            ;读出这个临时内存空间存储的数据
    BEQ     PRIOR1_99               ;如果为 0，检查优先级为 2 的任务
PRIOR1_05
    JSR     CHECKBIT0               ;否则，测试第 0 位
    BRSET   DO_TASK,SYSFLAG,PRIOR1_10  ; 如果第 0 位为 1，执行该任务
    BRA     PRIOR1_15               ;否则右移
PRIOR1_10
    JSR     WRITERAM                ;向 RAM 中写入子例程
    JSR     JUMPLONG                ;运行当前任务
    INC     ADD_POINTER             ;更新地址指针
    BCLR    DO_TASK,SYSFLAG         ;清除标志，表示任务已完成
PRIOR1_15
    JSR     SHIFTREG                ;右移临时寄存器
    LDA     SHADOWTASK,X            ;读临时寄存器
    BEQ     PRIOR1_99               ;如果寄存器为 0，检查优先级为 2 的任务
    JSR     INCSHIFT                ;否则，将移位计数器加 1
    LDA     SHIFTCNT,X              ;读移位计数器
    CMP     #$07                    ;是否已完成了最大次数的移位
    BHI     PRIOR1_99               ;如果是，检查优先级为 2 的任务
    BRA     PRIOR1_05               ;否则，测试临时寄存器的下一位
PRIOR1_99
    RTS
```

```
*****************************************************************
* 名称：PRIOR_ 2
* 功能：检查优先级为 2 的任务请求寄存器
*****************************************************************

PRIOR_2
    JSR     CLRSHIFT                ;将上一次的移位计数清零
    JSR     INC_LEVEL               ;增加优先级数
    LDA     SHIFTCNT,X              ;读出当前的移位计数
    BNE     PRIOR2_05               ;如果不等于 0,更新地址指针
    JSR     COPY                    ;拷贝任务请求寄存器到临时地址
PRIOR2_05
    JSR     UPDATE                  ; 更新地址指针
    ADD     #$10                    ;设置地址指针指向正确的地址
    STA     ADD_POINTER             ;
    LDX     PR_LEVEL                ;
    LDA     SHADOWTASK,X            ;读出任务请求寄存器的临时拷贝
    BNE     PRIOR2_99
PRIOR2_10
    BSET    TRY_PR3,SYSFLAG         ;设置标志,表示检查优先级为 3 的任务
PRIOR2_99
    RTS

*****************************************************************
* 名称：PRIOR_ 2OR3
* 功能：执行一个优先级为 2 或 3 的任务
*****************************************************************

PRIOR_2OR3
    BRSET   TRY_PR3,SYSFLAG,PRIOR23_99    ;如果 TRY_PR3 为 1,退出
    BRSET   GO_PR1,SYSFLAG,PRIOR23_20     ;如果 GO_ PR1 为 1,跳转
PRIOR23_05
    JSR     CHECKBIT0               ;否则测试寄存器的第 0 位
    BRSET   DO_TASK,SYSFLAG,PRIOR23_10    ;如果第 0 位为 1,执行任务
    JSR     SHIFTREG                ;否则,将寄存器右移
    JSR     INCSHIFT                ;增加移位计数
    BRA     PRIOR23_05              ;检查下一位
PRIOR23_10
    JSR     WRITERAM                ;向 RAM 中写入子例程
    JSR     JUMPLONG                ;执行任务
    BCLR    DO_TASK,SYSFLAG         ;清除标志,表示任务已完成
    JSR     SHIFTREG                ;右移临时寄存器
    LDA     SHADOWTASK,X            ;读临时寄存器
```

```
    BEQ     PRIOR23_15          ;如果寄存器为 0,跳转
    JSR     INCSHIFT            ; 否则,将移位计数器加 1
    LDA     SHIFTCNT,X          ;读移位计数器
    CMP     #$07                ;是否已完成了最大次数的移位
    BLS     PRIOR23_20          ;如果没有,跳转到 PRIOR23_ 15
PRIOR23_15
    JSR     CLRSHIFT            ;将移位计数清零
PRIOR23_20
    CLRA                        ;将地址指针复位
    STA     ADD_POINTER         ;指向优先级为 1 的任务地址
    BCLR    GO_PR1,SYSFLAG      ;清除标志,返回到对优先级 1 的处理
PRIOR23_99
    RTS

******************************************************************
* 名称: PRIOR_3
* 功能: 检查优先级为 3 的任务请求寄存器
******************************************************************

PRIOR_3
    JSR     INC_LEVEL           ;增加优先级数
    LDA     SHIFTCNT,X          ;读移位计数
    BNE     PRIOR3_05           ; 如果不等于 0,更新地址指针
    JSR     COPY                ; 拷贝任务请求寄存器到临时地址
PRIOR3_05
    JSR     UPDATE              ; 更新地址指针
    ADD     #$20                ; 设置地址指针指向正确的地址
    STA     ADD_POINTER
    BCLR    TRY_PR3,SYSFLAG     ;清除标志
    LDX     PR_LEVEL            ; 读出任务请求寄存器的临时拷贝
    LDA     SHADOWTASK,X        ;
    BEQ     PRIOR3_10           ;如果为 0,返回到对优先级 1 的处理
    BRA     PRIOR3_99           ;否则,检查第 0 位
PRIOR3_10
    BSET    GO_PR1,SYSFLAG      ;
PRIOR3_99
    RTS

******************************************************************
* 名称: WRITERAM
* 功能: 在 RAM 中写入一个子例程,使得调度器可以控制 16 位的地址,
*       也就是可以执行任务表中的任一任务。
******************************************************************
```

```
WRITERAM
    CLRH
    LDX     ADD_POINTER          ;读出任务表中的地址
    LDA     #$CD                 ;读"JSR"的机器码
    STA     JUMPLONG             ;复制到 RAM 中
    LDA     TASKTABLE,X          ;读出高 8 位地址
    STA     JUMPLONG+1           ;复制到 RAM 中
    AIX     #1                   ;增加地址
    STX     ADD_POINTER          ;
    LDA     TASKTABLE,X          ;读出低 8 位地址
    STA     JUMPLONG+2           ;复制到 RAM 中
    LDA     #$81                 ;读"RTS"的机器码
    STA     JUMPLONG+3           ;复制到 RAM 中
WRITERAM99
    RTS

*****************************************************************
* 名称：COPY
* 功能：复制任务请求寄存器
*****************************************************************

COPY
    CLRH
    LDX     PR_LEVEL             ;读任务寄存器
    LDA     TASKREQ,X            ;
    STA     SHADOWTASK,X         ;复制
    CLR     TASKREQ,X            ;将原寄存器清零
    RTS
*****************************************************************
* 名称：CHECKBIT0
* 功能：检查任务寄存器的第 0 位         *
*****************************************************************

CHECKBIT0
    CLRH
    LDX     PR_LEVEL             ;复制临时寄存器
    LDA     SHADOWTASK,X
    STA     TASKTEMP             ;
    BRSET   LSB,TASKTEMP,CHECK05    ;如果第 0 位为 1,执行任务
    INC     ADD_POINTER          ;否则更新地址指针
    INC     ADD_POINTER
    BRA     CHECK99              ;
CHECK05
    BSET    DO_TASK,SYSFLAG      ;设置标志,表示执行任务
```

```
CHECK99
    RTS

***************************************************************
* 名称：SHIFTREG
* 功能：将任务请求寄存器的临时拷贝向右移位
***************************************************************

SHIFTREG
    CLRH
    LDX     PR_LEVEL            ;逻辑右移
    LDA     SHADOWTASK,X        ;
    LSRA                        ;
    STA     SHADOWTASK,X        ;
    RTS

***************************************************************
* 名称：INCSHIFT
* 功能：增加当前优先级的移位计数器
***************************************************************

INCSHIFT
    CLRH
    LDX     PR_LEVEL            ;
    LDA     SHIFTCNT,X          ;读移位计数器
    INCA                        ;增加移位计数器
    STA     SHIFTCNT,X          ;
    RTS

***************************************************************
* 名称：CLRSHIFT
* 功能：将当前优先级的移位计数器清零
***************************************************************

CLRSHIFT
    CLRH
    LDX     PR_LEVEL            ;
    LDA     SHIFTCNT,X          ;
    CLRA                        ;
    STA     SHIFTCNT,X          ;
    RTS

***************************************************************
* 名称：INC_ LEVEL
* 功能：增加优先级数
***************************************************************
```

```
INC_LEVEL
    CLRH
    LDX      PR_LEVEL
    INCX                    ;
    STX      PR_LEVEL       ;
    RTS
******************************************************************
* 名称：UPDATE
* 功能：设置地址指针指向任务表的正确地址
******************************************************************
UPDATE
    CLRH
    LDX      PR_LEVEL       ;
    LDA      SHIFTCNT,X     ;
    LDX      #$02           ;
    MUL
    RTS
**************
* 任务表 *
**************
    ORG          TABLE
TASKTABLE
    FDB      TASKA
    FDB      DUMMY          ;没有使用的任务地址
    FDB      DUMMY
    FDB      TASKD
    FDB      DUMMY
    FDB      DUMMY
    FDB      TASKG
    FDB      DUMMY
    FDB      DUMMY
    FDB      DUMMY
    FDB      DUMMY
    FDB      TASKL
    FDB      DUMMY
    FDB      DUMMY
    FDB      DUMMY
    FDB      DUMMY
    FDB      DUMMY
    FDB      DUMMY
```

```
    FDB     DUMMY
    FDB     DUMMY
    FDB     TASKU
    FDB     DUMMY
    FDB     DUMMY
    FDB     TASKX
****************************************************************
* 任务代码 *
****************************************************************
DUMMY
    RTS                             ;空任务
TASKA
    LDA     #$01                    ;初始化
    STA     PORTA
    LDA     #$80
    STA     TIME_ON
TASKA_20
    JSR     DELAY                   ;延时
    DEC     TIME_ON                 ;减少 TIME_ ON
    LDA     TIME_ON                 ;读出 TIME_ ON
    BNE     TASKA_20                ;如果不等于 0,继续循环
    RTS

TASKD
    LDA     #$10                    ;初始化
TASKD_05
    STA     NUM_ON_LEDS             ;存储到 RAM 中
TASKD_10
    LDA     NUM_ON_LEDS             ;读出
    BNE     TASKD_12
    BSET    FINAL,APP_FLAG_REG      ;如果为 0,设置标志
    BRA     TASKD_15
TASKD_12
    DECA                            ;减少 LED 的显示数
TASKD_15
    STA     NUM_ON_LEDS             ;把数值复制回 RAM
    LSLA                            ;左移
    LSLA                            ;左移
    LSLA                            ;左移
    LSLA                            ;左移
    STA     PORTA                   ;把值发往端口 A
    LDA     #$25
```

```
    STA     TIME_ON
TASKD_20
    JSR     DELAY                    ;延时
    DEC     TIME_ON                  ;减少 TIME_ ON
    LDA     TIME_ON                  ;读出 TIME_ ON
    BNE     TASKD_20                 ; 如果不等于 0,继续循环
    BRSET   FINAL,APP_FLAG_REG,TASKD_99   ;如果标志为 1,退出
    BRA     TASKD_10                 ;否则,输出下一个数
TASKD_99
    BCLR    FINAL,APP_FLAG_REG       ;清除标志
    RTS
*****************************************************************
DELAY
    LDA     #$FF                     ;简单的延时函数
OUTLP
    DECA                             ;循环知道寄存器 A 为 0
    BNE     OUTLP
    INC     DELAY_VAR                ;增加计数器
    LDA     DELAY_VAR                ;读计数器
    CMP     #$CC                     ;是否等于十六进制的 CC
    BLS     DELAY                    ;如果不是,返回继续循环
DELAY99
    RTS
*****************************************************************
TASKG
    LDA     #$04                     ;初始化
    STA     PORTA
    LDA     #$80
    STA     TIME_ON
TASKG_20
    JSR     DELAY                    ;延时
    DEC     TIME_ON                  ;减少 TIME_ ON
    LDA     TIME_ON                  ;读出 TIME_ ON
    BNE     TASKG_20                 ;如果不等于 0,继续循环
    RTS
TASKL
    LDA     #$08                     ;初始化
    STA     PORTA
    LDA     #$80
    STA     TIME_ON
```

```
TASKL_20
    JSR     DELAY               ;延时
    DEC     TIME_ON             ;减少 TIME_ ON
    LDA     TIME_ON             ;读出 TIME_ ON
    BNE     TASKL_20            ;如果不等于 0,继续循环
    RTS

TASKU
    LDA     # $ 10              ;初始化
    STA     PORTA
    LDA     # $ 80
    STA     TIME_ON
TASKU_20
    JSR     DELAY               ;延时
    DEC     TIME_ON             ;减少 TIME_ ON
    LDA     TIME_ON             ;读出 TIME_ ON
    BNE     TASKU_20            ;如果不等于 0,继续循环
    RTS

TASKX
    LDA     # $ 20              ;初始化
    STA     PORTA
    LDA     # $ 80
    STA     TIME_ON
TASKX_20
    JSR     DELAY               ;延时
    DEC     TIME_ON             ;减少 TIME_ ON
    LDA     TIME_ON             ;读出 TIME_ ON
    BNE     TASKX_20            ;如果不等于 0,继续循环
    RTS

IRQ
    LDA     # $ 05              ;清除 IRQ 标志
    STA     $ 1D
    MOV     # $ 00,PORTA
    LDA     # $ 00
    STA     TASKREQ
    LDA     # $ 00
    STA     TASKREQ+1
    LDA     # $ 00
    STA     TASKREQ+2
    RTI
```

13.6 在 MC68HC08 CPU 上运行一个基于时间片的不可剥夺型实时内核

本内核主要利用微控制器内部的时钟模块产生恒定的定时中断，将 CPU 的运行时间分成了一个个的时间片。内核的任务调度工作都是在时间片的开始阶段完成的，而每个任务都必须在单个时间片内执行完成。

每个时间片都是从一次定时中断开始的，内核会在此时选择一个任务开始执行。在本内核中，任务的状态（就绪与否）并不重要，所有的任务不论处于什么状态，都会被内核定时调用。用户在编写任务程序时需要考虑到这一点，在任务的入口处自行判断任务的状态，并转入相应的处理。在这里，任务的优先级是唯一决定任务执行频度的因素，最高优先级的任务会在每两个时间片内被执行一次。

下面这个例子描述了具体的任务运行情况。

在这个例子中，每 0.5ms 发生一次时钟中断，而每 10 次时钟中断就组成了一个时间片（5ms）。任务 1 的优先级最高，每两个时间片（10ms）会执行一次。而任务 2 的优先级次之，每 4 个时间片（20ms）会执行一次。优先级最低的任务 6 则每 64 个时间片（320ms）会执行一次。

如前所述，每个任务的一次运行时间都不能超过一个时间片，用户不能编写运行时间

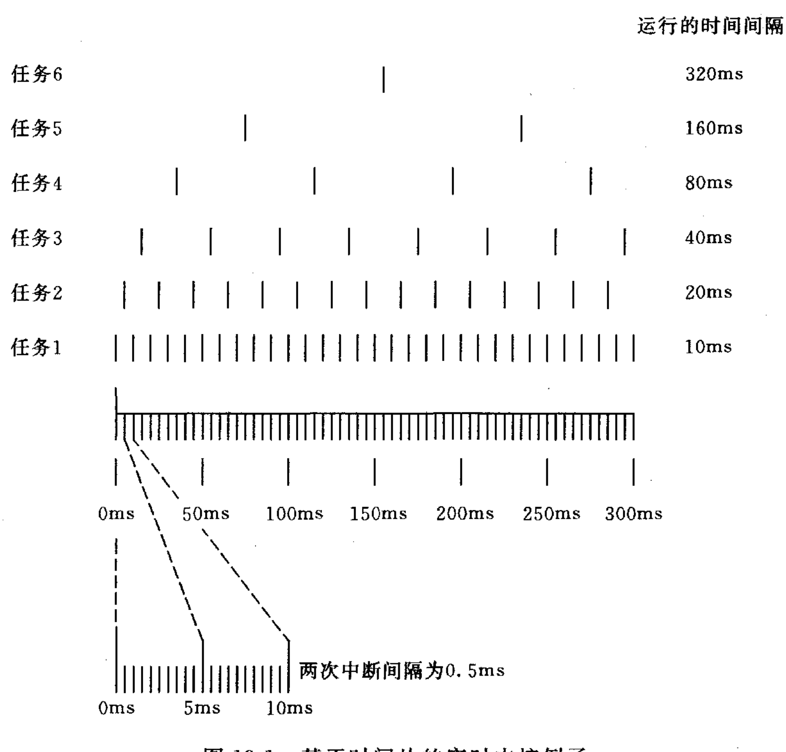


图 13-1 基于时间片的实时内核例子

过长的任务。如果用户需要进行较长时间的操作(例如对 Flash 编程),就必须将它分成几个子任务(例如擦除、写入、校验 3 个子任务)。

下面是一个在 MC68HC908GP32 上使用本内核的例子。这段代码是在 MC68HC908GP32IDK 开发系统上调试通过的,如果用户需要将它移植到其他的仿真器系统或是自己的目标系统,需要添加芯片的初始化代码和中断向量表。

```
**********************
* 端口定义 *
**********************
PORTA       EQU     $00           ;端口 A 数据寄存器
DDRA        EQU     $04           ;端口 A 方向寄存器
PORTB       EQU     $01           ;端口 B 数据寄存器
DDRB        EQU     $05           ;端口 B 方向寄存器
**********
* 内存 *
***********
ROM         EQU     $8000         ;ROM 地址
RAM         EQU     $B0           ;RAM 地址
VECTOR      EQU     $FFF6         ;向量表地址
***********************************
* 可编程时钟 *
***********************************
T1SC        EQU     $0020
T1MODH      EQU     $0023
T1MODL      EQU     $0024
*
********************************************************
* 用于决定任务时序的常量 *
********************************************************
TW_OCPER    EQU     $300          ;输出比较的周期
TW_TSPER    EQU     $08           ;时间片的周期(8 个输出比较的周期)
**************************
* 变量定义 *
**************************
     ORG    RAM
TV_TSCP     RMB     1             ;可编程时钟的时间片计数
TV_TSKCP    RMB     1             ;可编程时钟的任务计数
TV_TSKC     RMB     1
```

```
TV_DTASK  RMB   1
TV_STORE  RMB   1
TEMP      RMB   2
          ORG              ROM
*****************
* 主程序 *
*****************
T_SCHD05
    LDA     #$FF            ;设置端口 A 为输出状态
    STA     DDRA
    CLR     PORTA           ; 将端口 A 清零
    CLR     TV_TSKCP        ;将可编程时钟的任务计数器清零
    CLR     TEMP
    CLR     TEMP+1
T_SCHD99
    JMP     T_PROG05        ;
****************
*子程序*
****************
*****************************************************************
*名称:T_ PROG05
*功能:利用可编程时钟的输出比较功能实现任务执行的调度
*****************************************************************
T_PROG05
    LDA     #%01010000      ;允许时钟
    STA     T1SC
    LDHX    #TW_OCPER       ;每 5/8ms 发生一次时钟中断
    STHX    T1MODH
    CLR     TV_TSCP         ;将时间片计数器清零
PROG10
    CLI                     ;清除中断屏蔽位
PROG15
    BRSET   0,TV_DTASK,PROG20;如果第 0 位为 1,执行任务
    BRA     PROG15          ;否则等待下一次中断
PROG20
    LDHX    TEMP            ;增加任务间的间隔时间
    AIX     #1              ;
    STHX    TEMP            ;
    CPHX    #$108
    BNE     PROG30
```

```
    LDHX     # $00
    STHX     TEMP
    INC      TV_TSKCP             ;增加任务计数器
    JSR      T_TASK05             ;执行任务
PROG30
    BCLR     0,TV_DTASK           ;清除任务位
PROG99
    BRA      PROG10               ;等待下一次中断
******************************
* 中断服务例程 *
******************************
**************************************************
* 名称：T_ PRIN05
* 功能：检查是否需要执行某个任务，并更新输出比较寄存器
**************************************************
T_PRIN05
        PSHH
        LDA      T1SC
        BCLR     7,T1SC
*
        INC      TV_TSCP          ;增加时间片计数器
        LDA      TV_TSCP          ;读时间片计数器
        CMP      #TW_TSPER        ;比较
        BLO      PRIN99           ;如果 < 8,跳转到 T_ SCHED10
        CLR      TV_TSCP          ;如果 = 8,清除时间片计数器
        BSET     0,TV_DTASK       ;设置任务位
PRIN99
        PULH
        RTI                       ;返回
**************************************************
* 名称：T_ TASK05
* 功能：找出应该执行的任务并执行
**************************************************
****************
* 任务表 *
****************
T_TASK05
    LDA      TV_TSKCP
TASK15
    STA      TV_TSKC                  ;将任务存到 RAM 中
```

```
    BRCLR   0,TV_TSKC,TASK20        ;如果第 0 位为 0,执行任务 A
    BRCLR   1,TV_TSKC,TASK25        ;如果第 1 位为 0,执行任务 B
    BRCLR   2,TV_TSKC,TASK30        ;如果第 2 位为 0,执行任务 C
    BRCLR   3,TV_TSKC,TASK35        ;如果第 3 位为 0,执行任务 D
    BRCLR   4,TV_TSKC,TASK40        ;如果第 4 位为 0,执行任务 E
    BRCLR   5,TV_TSKC,TASK45        ;如果第 5 位为 0,执行任务 F
    BRCLR   6,TV_TSKC,TASK50        ;如果第 6 位为 0,执行任务 G
    BRCLR   7,TV_TSKC,TASK55        ;如果第 7 位为 0,执行任务 H
    CLRA                            ;
    STA     PORTA                   ;
    RTS                             ;返回
TASK20
    JSR     T_20                    ;跳转到第 1 个模块
    RTS
TASK25
    JSR     T_25                    ;跳转到第 2 个模块
    RTS
TASK30
    JSR     T_30                    ;跳转到第 3 个模块
    RTS
TASK35
    JSR     T_35                    ;跳转到第 4 个模块
    RTS
TASK40
    JSR     T_40                    ;跳转到第 5 个模块
    RTS
TASK45
    JSR     T_45                    ;跳转到第 6 个模块
    RTS
TASK50
    JSR     T_50                    ;跳转到第 7 个模块
    RTS
TASK55
    JSR     T_55                    ;跳转到第 8 个模块
    RTS
****************
* 任务 *
****************
T_20
    LDA     #$FE                    ;样本模块
    STA     PORTA
```

```
    RTS
T_25
    LDA     # $FD           ;样本模块
    STA     PORTA
    RTS
T_30
    LDA     # $FB           ;样本模块
    STA     PORTA
    RTS
T_35
    LDA     # $F7           ;样本模块
    STA     PORTA
    RTS
T_40
    LDA     # $EF           ;样本模块
    STA     PORTA
    RTS
T_45
    LDA     # $DF           ;样本模块
    STA     PORTA
    RTS
T_50
    LDA     # $BF           ;样本模块
    STA     PORTA
    RTS
T_55
    LDA     # $7F           ;样本模块
    STA     PORTA
    RTS
```

13.7 在 MC68HC08 CPU 上运行可剥夺型实时内核 μC/OS-Ⅱ

可剥夺型实时内核 μC/OS-Ⅱ的前身是 μC/OS。μC/OS-Ⅱ在 μC/OS 的基础上作了一些改进，增加了一些新功能。这个实时内核已经被广泛地用在照相机业、医疗器械、音像设备、发动机控制、网络接入设备、高速公路电话管理系统、自动提款机、工业机器人等方面。

μC/OS 和 μC/OS-Ⅱ在性能上并不亚于商业级软件的 RTOS。区别在于 μC/OS 只是一个实时操作系统内核。而商业软件一般是一个包括调试手段在内的完整的软件包。μC/OS 和 μC/OS-Ⅱ可以在 PC 的 DOS 环境下运行。95%以上代码用 C 语言写成，而与

CPU 硬件相关的代码只有两个文件，每个文件也只有约两页纸。其中一个文件用汇编语言写成，另一个文件用 C 语言写成。μC/OS 和 μC/OS-Ⅱ是专门为嵌入式应用写的实时内核，除可移植性外，可固化、可裁剪。用户可以只选用对其应用程序有用的那一部分，故内核目标代码可以剪裁到小于 1.5kB，大则可以大到 10kB 以上。μC/OS 和 μC/OS-Ⅱ可以管理和调度约 60 个任务。由于 μC/OS 和 μC/OS-Ⅱ是可剥夺型的，在任务调度过程中要处理可能发生的竞争，每个任务都要有自己的栈空间，故任务越多，RAM 的需求量就越大。

最初的 μC/OS 是为 MC68HC11 工作在扩展方式下写的。MC68HC11 是增强型 8 位 CPU，ROM 和 RAM 总共的寻址空间为 64kB，一般设计成 8kB RAM 和 48kB ROM。

MC68HC908GP32 有 32kB Flash，也就是说程序空间有 32kB，拿出几千字节来运行实时内核没有问题。但在 MC68HC08 中，运行 μC/OS 和 μC/OS-Ⅱ的主要困难是 RAM 只有 512B，运行可剥夺型实时内核显得太少，只能把中断嵌套层次限制得很低，任务数目也只能定义为很少的几个，只使用实时内核中几个主要的功能函数。因为这些宝贵的 RAM 资源被系统和实时内核占用，留给应用程序用的 RAM 空间就变得十分有限，故本书中给出在 MC68HC08 上运行的可剥夺型内核 μC/OS-Ⅱ的主要目的是告诉读者如何写和 CPU 硬件有关的程序，而利用 C 语言写的源代码可以直接拿来用。可剥夺型实时内核更适用于 RAM 空间较大的系统中，至少应有几千字节 RAM 可用才好。

要实现 μC/OS-II 向 MC68HC908GP32 的移植，需要做两方面的工作：一是重新定义内核的大小和功能；二是为内核编写与硬件相关的代码。下面以一个实例讲述如何在 MC68HC908GP32 上实现 μC/OS-II。本例使用的 C 语言编译器由 HIWARE 公司提供。

13.7.1 重新定义内核的大小和功能

首先是编写整个项目的公共头文件 INCLUDES. H，这个文件会被所有的 C 语言源程序引用。在本例中此文件的代码如下：

```
/ ******************************************************************
*                INCLUDES. H
****************************************************************** /

#include    "OS_CFG. H"
#include    "OS_CPU. H"
#include    "UCOS_Ⅱ. H"
```

这 3 个头文件必须被引用，如果用户在自己的项目中编写了其他的头文件，也可以在这里引用。

需要编写的头文件还有 OS_CFG. H，这个文件用于配置内核的属性。在本例中此文件的代码如下：

```
/ ******************************************************************
```

```
*                    OS_CFG. H
*               与配置相关的代码
****************************************************************** /
/* 系统中最大的事件控制块的数量 */
#define OS_MAX_EVENTS             2
/* 系统中最大的内存块数 */
#define OS_MAX_MEM_PART           2
/* 系统中最大的消息队列数 */
#define OS_MAX_QS                 2
/* 用户程序中最大的任务数 */
#define OS_MAX_TASKS              4
/* 系统中的任务最低优先级(最大优先级数) */
#define OS_LOWEST_PRIO            20
/* μC/OS-Ⅱ中空闲任务(idle task)堆栈的容量 */
#define OS_TASK_IDLE_STK_SIZE     60
/* 系统不使用μC/OS-Ⅱ中的统计任务(statistic task)及其初始化函数 */
#define OS_TASK_STAT_EN           0
/* μC/OS-Ⅱ中统计任务堆栈的容量 */
#define OS_TASK_STAT_STK_SIZE     20
/* 在文件 OS_CPU_C. C 中声明对外接口函数(hook function) */
#define OS_CPU_HOOKS_EN           1
/* 不使用μC/OS-Ⅱ中的消息邮箱函数及其相关数据结构 */
#define OS_MBOX_EN                0
/* 不使用μC/OS-Ⅱ中的内存块管理函数及其相关数据结构 */
#define OS_MEM_EN                 0
/* 不使用μC/OS-Ⅱ中的消息队列函数及其相关数据结构 */
#define OS_Q_EN                   0
/* 使用μC/OS-Ⅱ中的信号量管理函数及其相关数据结构 */
#define OS_SEM_EN                 1
/* 不使用μC/OS-Ⅱ中的 OSTaskChangePrio( )函数 */
#define OS_TASK_CHANGE_PRIO_EN 0
/* 使用μC/OS-Ⅱ中的 OSTaskCreate( )函数 */
#define OS_TASK_CREATE_EN         1
/* 不使用μC/OS-Ⅱ中的 OSTaskCreateExt( )函数 */
#define OS_TASK_CREATE_EXT_EN     0
/* 不使用μC/OS-Ⅱ中的 OSTaskDel( )函数 */
#define OS_TASK_DEL_EN            0
/* 不使用μC/OS-Ⅱ中的 OSTaskSuspend( )和 OSTaskResume( )函数 */
#define OS_TASK_SUSPEND_EN        0
/* 调用 OSTimeTick( )函数的频率 */
#define OS_TICKS_PER_SEC          10
```

由于 MC68HC908GP32 内部的 RAM 有限，所以本例尽量减少了内核中不必要的模块（包括邮箱、消息队列、内存管理等），去掉了任务挂起、唤醒和删除等扩展功能，但仍然支持任务的创建和管理，也保留了信号量模块来用于任务间的通信。

13.7.2 修改内核代码的汇编部分

为内核编写与硬件相关的代码可以用 C 语言，也可以用汇编语言。虽然 C 语言的编写较为方便，但为了提高内核的速度和效率，建议用户使用汇编语言编写任务调度部分的代码。

第一个文件是 OS_CPU.H，用于设置与微控制器的 CPU 核心相关的属性，包括各种数据类型对应的存储长度等等，具体的代码如下：

```
/*************************************************************
*             OS_CPU.H
*             与 6808 相关的代码
*************************************************************

/*数据类型*/
#ifndef    _TYPES
typedef    unsigned char     BOOLEAN;
typedef    unsigned char     INT8U;
typedef    signed char       INT8S;
typedef    unsigned short    INT16U;
typedef    signed short      INT16S;
typedef    unsigned long     INT32U;
typedef    signed long       INT32S;
#define    _TYPES
#endif

#define    OS_STK_GROWTH   1        /* 堆栈从高地址向低地址增长 */
#define    OS_STK                   INT8U

#define     OS_ENTER_CRITICAL( )     asm sei
#define     OS_EXIT_CRITICAL( )      asm cli
#define     OS_TASK_SW( )            asm swi
```

另一个文件是 OS_CPU_C.C，含有为内核编写的底层函数。这些函数分为三类：一是任务堆栈的初始化(OSTaskStkInit)；二是用户的接口(HOOK)函数，利用这些函数，用户可以在内核运行时插入自己的处理；三是任务切换时的底层操作函数。这些函数需要直接操作内存和堆栈，是与微控制器硬件体系关系最密切的部分。具体的代码如下：

```
/*************************************************************
*               OS_CPU_C.C
```

```
*************************************************************/
#define   OS_CPU_GLOBALS
#include "includes.h"

/*************************************************************
*               初始化任务堆栈
/*************************************************************

void *OSTaskStkInit (void (*task)(void *pd), void *pdata, void *ptos, INT16U opt)
{
    INT16U *stk;
    stk = (INT16U *)ptos;                  /* 读出堆栈指针 */
    *--stk = (INT16U)(task);               /* 程序计数器 */
    *--stk = (INT16U)(0x00);               /* X和A寄存器 */
    *--stk = (INT16U)(0x00);               /* CCR和H寄存器 */
    return ((void *)stk);
}

#if OS_CPU_HOOKS_EN

/*****************************************
*                  任务建立接口函数
*****************************************/

void OSTaskCreateHook (OS_TCB *ptcb)
{
    ptcb = ptcb;            /* 避免编译器警告 */
}
/*****************************************
*            任务删除接口函数
*****************************************/
void OSTaskDelHook (OS_TCB *ptcb)
{
    ptcb = ptcb;             /* 避免编译器警告 */
}
/*****************************************
*            任务切换接口函数
*****************************************/
void OSTaskSwHook (void)
{
}
/*****************************************
*            统计任务接口函数
*****************************************/
void OSTaskStatHook (void)
```

```
{
}
/*****************************************
*            时钟节拍接口函数
*****************************************/
void OSTimeTickHook (void)
{
}
#endif
/*****************************************
* 启动优先级最高的任务
*****************************************/
void OSStartHighRdy(void)
{
asm
{
    JSR     OSTaskSwHook            // 调用用户定义的接口函数
    LDA     OSRunning               // 标识多任务状态
    INCA
    STA     OSRunning
    LDX     OSTCBHighRdy            // 指向优先级最高的就绪任务的 TCB
    STX     OSTCBCur
    PSHX
    LDX     OSTCBHighRdy:1
    STX     OSTCBCur:1
    PULH
    LDA     0,x                     // 读出 SP
    PSHA
    LDX     1,x
    PULH
    TXS
    PULH                            // 恢复 H 寄存器
    RTI                             // 运行任务
}
}
/*****************************************
* 中断级的任务切换
*****************************************/
void OSIntCtxSw(void)
{
asm
```

```
{
    AIS      #4
    TSX
    PSHX
    PSHH
    LDX      OSTCBCur                    // 指向当前任务的 TCB
    PSHX
    LDX      OSTCBCur:1
    PULH
    PULA
    STA      0,x                         // 保存堆栈指针到 TCB
    PULA
    STA      1,x
    JSR      OSTaskSwHook                // 调用用户定义的接口函数
    LDA      OSPrioHighRdy               // OSPrioCur = OSPrioHighRdy
    STA      OSPrioCur
    LDX      OSTCBHighRdy                // 指向优先级最高的就绪任务的 TCB
    STX      OSTCBCur
    PSHX
    LDX      OSTCBHighRdy:1
    STX      OSTCBCur:1
    PULH
    LDA      0,x                         // 读出 SP
    PSHA
    LDX      1,x
    PULH
    TXS
    PULH                                 // 恢复 H 寄存器
    RTI                                  // 执行任务
}
}
/*******************************************
*           任务级的任务切换
*******************************************/
void OSCtxSw(void)
{
asm
{
    PSHH                                 // 保存 H 寄存器
    TSX
    PSHX
```

```
    PSHH
    LDX      OSTCBCur                 // 指向当前任务的 TCB
    PSHX
    LDX      OSTCBCur:1
    PULH
    PULA
    STA      0,x                      // 保存堆栈指针到 TCB
    PULA
    STA      1,x
    JSR      OSTaskSwHook             // 调用用户定义的接口函数
    LDA      OSPrioHighRdy            // OSPrioCur = OSPrioHighRdy
    STA      OSPrioCur
    LDX      OSTCBHighRdy             // 指向优先级最高的就绪任务的 TCB
    STX      OSTCBCur
    PSHX
    LDX      OSTCBHighRdy:1
    STX      OSTCBCur:1
    PULH
    LDA      0,x                      // 读 SP
    PSHA
    LDX      1,x
    PULH
    TXS
    PULH                              // 恢复 H 寄存器
    RTI                               // 执行任务
}
}
```

最后还需要编写的是微控制器的初始化代码和时钟中断服务程序。微控制器的初始化代码除了与具体系统结构有关以外，还与具体的外围硬件电路有关，是系统能否正常工作的前提条件，但由于它与 μC/OS-Ⅱ并没有直接的联系，在这里就不再赘述了。

μC/OS-Ⅱ要求微控制器提供一个简单的时钟，用于任务的延时等功能。在这里，我们利用 MC68HC908GP32 的 16 位定时器的溢出中断实现了一个 10Hz 的时钟，其初始化代码如下：

```
LDA      #0x50      // 允许定时器
STA      T1SC
LDHX     #0x0333    //100ms 发生一次时钟中断
STHX     T1MODH
CLI
```

在溢出中断的服务程序中，有 3 个内核函数必须被调用：OSIntEnter( )、

OSTimeTick( )和 OSIntExit( ),具体的代码如下：

```
void TimerISR(void){
asm{
      PSHH
      LDA       T1SC
      BCLR      7,T1SC
}
OSIntEnter( );          /* 提示 μC/OS-Ⅱ已进入中断服务例程 */
OSTimeTick( );          /* 调用 μC/OS-Ⅱ的时钟节拍函数 */
OSIntExit( );           /* 提示 μC/OS-Ⅱ中断服务例程结束 */

asm{
     PULH
     RTI
}
}
```

13.7.3 编写用户应用程序

在为内核编写了上述内存管理及与硬件相关的代码以后,用户就可以为自己的项目编写实际的代码了。在本例中,用户任务共有 2 个。任务 1 在初始化了时钟中断以后,就进入了一个死循环。在这个循环里,任务 1 一方面以 1s 为周期改变并行 I/O 口 PORTA 第 0 个引脚的输出电压,另一方面每隔 4s 便向任务 2 发送一个信号。而任务 2 则始终等待任务 1 发来的信号,一旦收到信号,便改变并行 I/O 口 PORTA 第 1 个引脚的输出电压。具体的代码如下：

```
/*********************************************
 * EXE2.C
*********************************************/
#include <hidef.h>
#include "includes.h"
#if defined(__HC11__)
    #pragma DATA_SEG SHORT zpage /* 以下的变量会放入第 0 页内存中 */
#endif
Byte     PORTA     @0x0000;
Byte     DDRA      @0x0004;
Byte     T1SC      @0x0020;
Byte     T1MODH    @0x0023;
OS_EVENT  *Semaphore;
```

```
/** 应用程序的堆栈变量 **/
#define TASK_STK_SIZE   45

INT8U   Task1Stk[TASK_STK_SIZE];
INT8U · Task2Stk[TASK_STK_SIZE];

void HardwareInit(void);
/************************************************************************/

/**   各任务代码                    **/

void Task1(void * pdata)
{
int count=0;
/* 时钟初始化 */
asm{
      LDA     #0x50        //允许定时器
      STA     T1SC
      LDHX    #0x0333      //100ms 发生一次中断
      STHX    T1MODH
      CLI
}
    for(;;){
    PORTA&=0xFE;
    /* 延时 0.5 s */
    OSTimeDly(5);
    PORTA|=0x01;
    /* 延时 0.5s */
    OSTimeDly(5);

    count++;
    if(count==4){
        OSSemPost(Semaphore);
        count=0;
    }
    }
}

void Task2(void * pdata)
{
   Byte err;
   for(;;){
      OSSemPend(Semaphore, 0, &err);
   PORTA&=0xFD;
      OSSemPend(Semaphore, 0, &err);
   PORTA|=0x02;
```

```
    }
}
void main(void){
/* 初始化硬件 */
HardwareInit( );
/* 初始化 OS,建立任务并启动 OS */
OSInit( );
Semaphore = OSSemCreate(0);
OSTaskCreate(Task1, (void *)0, (void *)&Task1Stk[TASK_STK_SIZE], 10);
OSTaskCreate(Task2, (void *)0, (void *)&Task2Stk[TASK_STK_SIZE], 9);
OSStart( );
}
```

在主程序中,用户必须先调用 OSInit( ),然后创建各个任务和信号量等,最后调用 OSStart()以启动内核运行,开始正常的任务调度。

第 14 章　设计应用举例

14.1　智能电话门铃

这个例子使用 MC68HC908GP32 来实现一个可以智能联络主人的门铃。需要实现的功能有：

(1) 有人拜访时，自动呼叫设定的电话；

(2) 主人可以和来访者进行交谈；

(3) 访客留言；

(4) 可以通过电话遥控开门。

14.1.1　硬件设计

为了完成上述功能，还需要一些其他的芯片器件。这些芯片可以完成特定的功能，而且这些芯片都有微控制器的控制接口。在设计硬件结构的时候，要做好安排，充分利用 MC68HC908GP32 的片上资源。

首先完成这个例子的硬件结构设计，如图 14-1 所示。

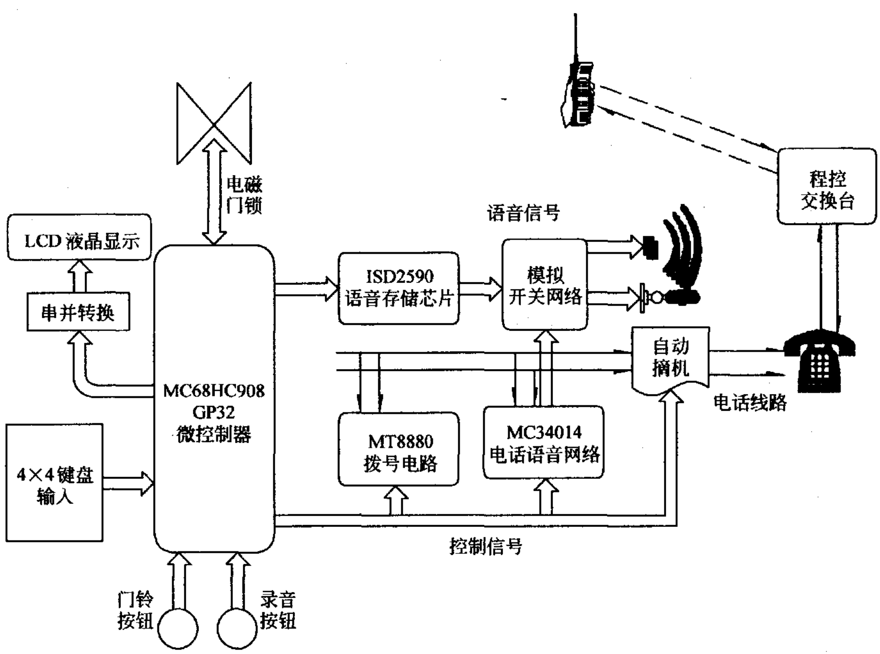


图 14-1　系统框图

下面介绍例子中用到的各种芯片，重点放在如何将这些芯片和微控制器连接。

(1) MC34014 是一片电话语音网络集成电路，完成和电话线路的接口。可以通过微控制器的 I/O 口来控制 MC34014 的一些功能，如静音等。但在这个例子中，MC34014 和微控制器没有直接的连接控制。

(2) MT8880 是一个完整的多音双频发送接收器，用来完成电话拨号和按键检测。它需要微控制器来控制，通过写入内部的控制寄存器可以确定工作方式，将电话号码依次写入数据寄存器可以完成拨号流程。控制的接口包括 4 位数据线、4 位控制线(实现读写时序)和信号输出端 IRQ(输出滤波后的电话信号波形)。在设计时，使用 MC68HC908GP32 的并行 I/O 口的 B 口的低 4 位作为 MT8880 的双向数据线，高 4 位作为 MT8880 的控制线。程序必须通过改变控制线的输出电平模拟出 MT8880 的读写时序，这就需要每一根控制线都可以单独操作，而 MC68HC908GP32 的并行 I/O 口可以通过位操作满足这一点。数据线是双向的，而控制线是单向的，因为并行 I/O 口是双向的，并且每一位都可以设定方向，所以这里数据和控制线分享 B 口不会引起冲突。之所以设计成数据和控制线分享 B 口，一是因为这样比较明确，B 口就是完全和 MT8880 联系的，便于程序出现问题的时候确定问题所在；二是因为这样使得 MT8880 和微控制器的连线紧凑，便于布线。MT8880 的呼叫模式可以将电话线的呼叫信号滤波送到 IRQ 引脚，通过测量信号的频率和间隔，可以得知电话线路是否空闲、呼叫是否接通等，所以将该信号引脚接到 MC68HC908GP32 的 PD4 口，利用该引脚的输入捕捉功能来检查信号频率和间隔(可以参考定时器章节的程序)。

(3) ISD2500 是语音芯片，可以录制和重播 90s 的语音，在这个例子中用来实现语音提示和访客留言。该芯片带微处理器接口，3 根控制线用来确定芯片的工作方式(录音或播音)、播放、录音或暂停，设计时使用 MC68HC908GP32 的并行 I/O 口的 A 口的低 3 位作为控制线。10 根地址线提供了多达 600 段的语音片。通过合适的选址，可以划分出确定时间的语音片断。在设计时将低 5 位地址接地，高 5 位和并行 I/O 口的 A 口的高 5 位连接，用来确定 32 段语音。当分段的语音播放结束后，ISD2500 从 EOM 引脚送出一个脉冲，为了捕捉这个脉冲，将这个引脚和 MC68HC908GP32 的 PD5 口连接，利用这个口的输入捕捉功能。

(4) 在这个例子中，语音信号往往要分路传送。例如语音芯片的存储语音有时需要传输到电话网络中，有时需要传输到外部扬声器，而有时还需要接受外部话筒的输入语音信号。所以必须使用模拟开关根据不同时刻的需要来切换这些语音通道。HEF4053 是 CMOS 的二通道模拟开关，通过 2 个 HEF4053 的连接控制，切换 MC34014、ISD2590 以及扬声器和外部话筒之间的语音通道。一共使用了 2 片 HEF4053 提供了 4 种切换模式。其中录音时的通道控制可以使用 ISD2500 的录音控制线。在这个例子里，没有使用 MC68HC908GP32 的 SCI 异步串行通信功能，所以将 PE0 和 PE1 用作另外两根通道控制线。

使用这些芯片，可以完成例子所提出的大部分功能，但还需要一些其他的器件：

(1) 使用 4×4 的键盘作为输入接口，提供电话拨号盘的“0～9”和“#”“*”键，另外还可以提供“确定”、“取消”、“设置”等功能键。键盘的接口需要 8 位 I/O，而现在

MC68HC908GP32 的 8 位并行 I/O 口的 A 口和 B 口分别用于 ISD2590 和 MT8880 的控制，没有剩余的键盘输入口。可以注意到，A 口对 ISD2590 的控制全部是输出控制，不需要输入信号。所以通过输入输出复用，可以使用 A 口的输入来接受键盘信号，输出用来控制语音芯片 ISD2590。ISD2590 的控制信号比较复杂，有时序的要求，而键盘的输入信号只是简单的电平变化，复用不会引起 ISD2590 的误动作。另外 PORTA 口是键盘中断口，正好适合用来连接键盘。

(2) 这个例子还设计了使用一块字符型的液晶显示信息。液晶是并行接口，需要并行送入地址信息。但现在 A 口已经用于键盘输入和输出控制 ISD2590，B 口用于 MT8880 的输出控制和双向的数据传递。所以只有使用 MC68HC908GP32 的 SPI 同步串行口即 D 口，通过一片串并转换的 74LS164 来传递液晶的地址信号。74LS164 和液晶还需要 4 根控制信号，使用并行 I/O 口剩下的 C 口的高 4 位。

(3) 外部中断(IRQ)连接门铃按钮，当有客人访问按动门铃触发外中断。在结束一位客人的访问请求后，系统会进入低功耗的停止状态，当有外部中断的时候，又会从停止状态恢复出来。另外还需要一根控制线来实现电话摘机，使用 PC0。

(4) 现在 MC68HC908GP32 所有的引脚都已经被占用，但是还有录音按钮的输入信号和电磁门锁的输出控制信号还没有实现。分析一下，虽然有很多口只是作为输出控制(PC0～PC4，PE0，PE1)，但如果复用这些口容易导致按钮的输入信号使得原来输出控制的对象误操作，不可靠。注意到键盘的输入信号是有一定的规律的，每一个按键必然导致 A 口的两个位的电平变化，对应这个键的行列位置，而录音按钮只是一位的电平变化。所以通过程序的判断，可以使用 A 口的任一位来实现录音按钮的输入。电磁门锁的输出控制可以通过 PD5 的复用来实现，考虑到 PD5 的输入信号的影响，简单的复用容易引起电磁门锁的误操作。从 ISD2590 的手册可以知道，语音播放结束的脉冲信号是非常窄的信号，那么通过一个简单的积分和整形电路就可以去掉这个窄脉冲。PD5 用作输出控制电磁门锁时，只要送出一定宽度的脉冲就可以满足积分电路而使得电磁门锁动作，而 ISD2590 的语音结束的脉冲对电磁门锁没有影响。

现在，例子的硬件设计完成了，所有要求的功能在硬件上都能够实现，但有些功能还需要程序的配合。这个例子中，MC68HC908GP32 的功能被充分地使用，所有的 I/O 口都被合理的分配和使用。充分研究这个例子可以进一步熟悉 MC68HC908GP32 芯片的功能和应用方法。

14.1.2 软件流程

这里只给出这个例子的工作流程，如图 14-2 所示。具体程序根据流程很容易编写。

(1) 来访者按动门铃，呼叫器工作，通知室内主人。

(2) 在一定时间后主人没有开门，认为主人不在，通知客人稍候并开始呼叫主人。

(3) 若主人接机，接通语音网络，主人可以和来访者交谈。主人通过电话输入确认密码，可以选择远程开门。

(4) 若主人在一定时间内没有接机，则通知客人主人不在，并请客人留言。

(5) 结束一个客人的访问处理，进入低功耗状态。

实现这些流程已经充分地运用了所设计的硬件的大部分功能。如果在这个基础上继续开发，还可以实现更多的功能。由于电路具备了自动拨号功能，并且有功能较强的微控制器，所以通过编程可以实现电话簿、来电转接、定时拨号、电话使用纪录、电话防盗、来电显示等功能。

通过外接一些监控传感器，可以监测敏感或需要保密的地方，发现险情或盗情，可以通过电话线自动报警。该电路已经具有了多音双频信号识别功能，在此基础上可以开发出更多的功能。例如，当拨通电话以后，通过输入多音双频信号实现交互，进行身份确认，执行远程控制操作等。当然，这些功能不是每一个用户都需要的。此外，每项功能都会占用一定的系统资源而影响到其他功能的效率。针对特定用户的特定需求，通过软件的更改去掉不必要的功能，将资源分配给用户的特殊需求。对以前的掩膜或是EPROM 等芯片来说这是不可能的，因为对这些芯片来说程序的写入和修改是不可逆或非常麻烦的。但对于 MC68HC08GP32 来说，这确实是可以实现的，32kB 的 Flash 存储器可以非常方便地动态更改程序。设计一个通用的、适应性好的硬件平台，通过软件的调整适应不同场合的需要，这为微控制器系统的开发和嵌入式应用提出了更广阔的方向。

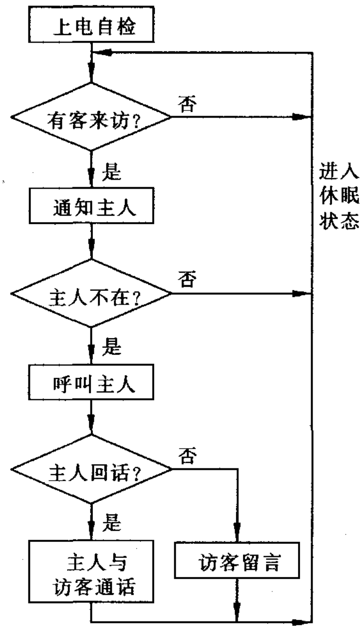


图 14-2　软件流程图

14.2　MC68HC908GP32 在智能 UPS 电源中的应用

MC68HC908GP32 内部采用了锁相环电路，可以产生最大 8MHz 的内部时钟，处理速度比一般的 8 位机有了很大的提高，这样一些一般 8 位机需用专门硬件完成的任务在 MC68HC908GP32 上可由软件完成，降低了系统的成本。除了速度上的优势外，MC68HC908GP32 内部还有 32kB 的 Flash 存储器，可以用软件进行擦写，能够保存程序运行中的数据，例如用户设置的信息等等。本节将介绍 MC68HC908GP32 的这些特点在智能 UPS 电源上的应用。UPS 电源系统的许多概念属于电力电子学领域，为了便于读者理解，本节只作必要的原理性介绍。

UPS 电源的工作原理简述如下：电源启动后，将不断检测电网的状态，如果电网工作不正常，例如出现断电、电压过低、电压不稳等，UPS 将启动逆变器，将蓄电池的电能转变为 220V，50Hz 的输出供负载使用，直到电网恢复正常后，关闭逆变器。电源的持续工作时间取决于所带电池的容量。

按功能划分，UPS 电源分为逆变器、电网检测、人机界面、通信等模块，都与作为控制核心的微控制器有密切关系。逆变器负责将 12V 蓄电池的电能转变为 220V，50Hz 的输

出供负载使用，其原理是使用功率开关器件和高频变压器产生振荡升压。驱动功率开关器件的信号由微控制器产生。改变驱动信号的频率、占空比，就可以控制逆变器，调节输出电压。关闭驱动信号，也就关闭了逆变器。为了能保证输出电压的稳定，对驱动信号有很高的要求。电网检测电路负责检测电网状态，判断是否有断电、电压抖动等等。电网检测电路将向微控制器输出一个逻辑信号，反映电网是否正常，微控制器将根据这一信号决定是否启动逆变器等一系列操作。人机界面负责用户输入和显示，用户用键盘输入，键盘状态由微控制器的 I/O 口读入。显示输出为 LCD 模块，由微控制器的 I/O 口直接驱动。通信模块通过 RS232 串行口与 PC 通信，用户可以用 PC 上的监控软件查看电源状态，设置电源参数等等。使用监控软件将为用户提供更友好的人机界面。

14.2.1 硬件结构

UPS 电源的结构如图 14-3 所示。

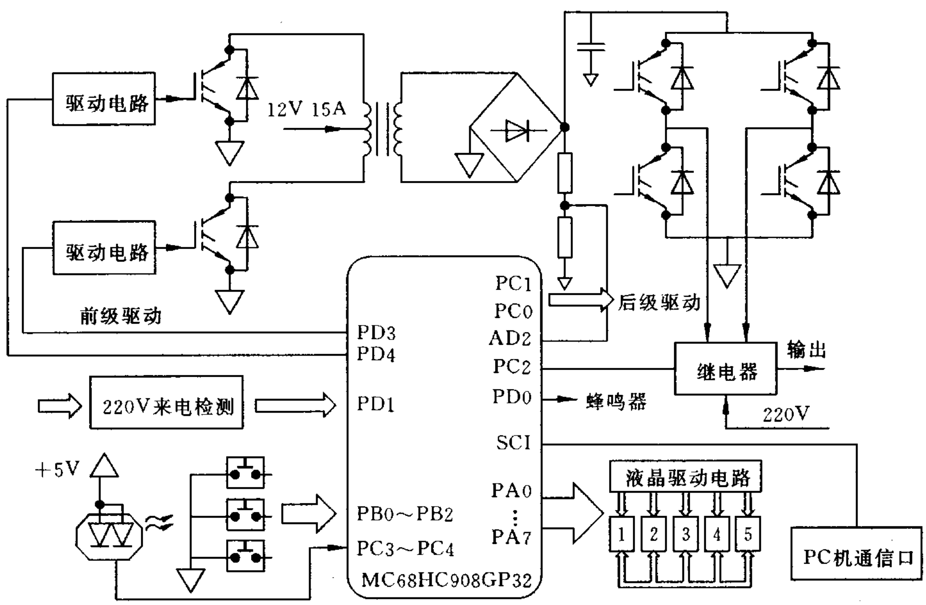


图 14-3 以 MC68HC908GP32 为核心的 UPS 系统示意图

在 UPS 系统中，逆变器、电网检测等电路工作在高电压、大电流状态，需要与微控制器部分隔离。由于微控制器产生的驱动信号为 0～5V，不能直接驱动逆变器电路的功率器件，所以需要驱动电路对微控制器输出的信号进行电平放大。

在 UPS 电源系统中，MC68HC908GP32 主要完成下述功能：

(1) 产生驱动信号。在高频 UPS 电源中，逆变器的工作需要两组，共 4 路驱动信号。一组的工作频率为 40 kHz，用于逆变器前端的推挽升压电路(图 14-4)，将 12V 蓄电池升压；另一组的工作频率为 50Hz，用于后端的桥式变换电路(图 14-5)。

对于 40 kHz 的一组信号，一般的 8 位微控制器工作频率较低，不能直接产生，而通常

采用一片专用的信号发生芯片。采用 MC68HC908GP32 后，外接 32.768 kHz 晶振，在 8MHz 的总线时钟下，可以用软件产生驱动信号，从而降低了硬件的成本。推挽升压电路如图 14-4 所示，由 MC68HC908GP32 的 PD3、PD4 引脚输出的驱动信号，经过驱动电路隔离和放大后驱动两路开关器件。PD3、PD4 产生的信号相位差为 90°，目的是控制两路开关器件轮流导通和截止，与变压器一起完成推挽升压的过程。由此可见，PD3 和 PD4 的信号在相位同步上有严格的要求，需要用软件精确控制。如果不同步，将损坏开关器件。

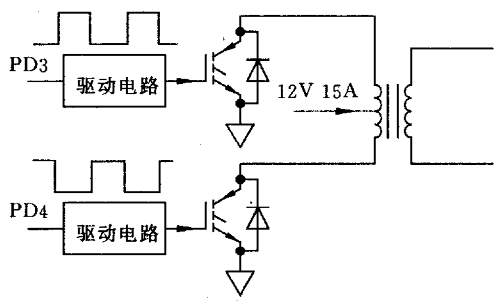


图 14-4 推挽升压电路示意

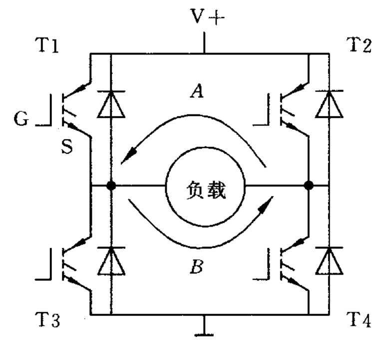


图 14-5 全桥变换电路示意

推挽升压电路的输出经过整流滤波后得到一个 300V 左右的高压直流，再经过桥式电路（如图 14-5），变换为负载所需的 220V，50Hz 的交流输出。桥式电路是电力电子设备中广泛使用的一种变换电路。基本的桥式电路由 4 个开关器件组成。同一组的 T1 和 T3 或 T2 和 T4 构成一个桥臂。正常工作时 4 个开关器件交替导通和截止。例如，某一时刻，T2 和 T3 导通，T1 和 T4 截止，此时电流沿 A 箭头方向流过负载。在下一时刻，T1 和 T4 导通，T2 和 T3 截止，电流将沿箭头 B 方向流过负载。重复这一过程，在负载上就可得到方向交替改变的交流电。控制开关器件关断和导通的信号由 MC68HC908GP32 的 PC0 和 PC1 输出，这一组信号将控制 220V 输出的频率和波形。如果驱动信号的占空比按正弦规律变化，则桥式电路的输出经过滤波后，在负载上就可得到正弦交流电。需要注意的是，桥式电路工作在高电压状态，需要与来自 MC68HC908GP32 的驱动信号隔离，这也是由驱动电路完成的。关于桥式电路以及驱动电路，感兴趣的读者可以参考有关电力电子学方面的资料。

本例中，利用 MC68HC908GP32 的定时器中断产生两组驱动信号，分别从 I/O 引脚（PD3～PD4、PC0～PC1）输出，驱动推挽升压电路和桥式变换电路。当 PD3～PD4 和 PC0～PC1 都为低电平时，将关闭推挽升压电路和桥式变换电路。

（2）220V 来电检测。此部分电路检测 220V 电网状态，如果电网不正常，则立即启动逆变器，向负载供电。本例中用 MC68HC908GP32 的 PD1 引脚作为逻辑输入，在程序中用软件判断引脚状态。需要注意的是，由于电网电压不稳，PD1 的输入信号可能会有抖动。在软件设计中要采取相应的措施，例如延时判断，确信信号稳定后再执行相应操作。

（3）输出切换。MC68HC908GP32 的引脚 PC2 控制继电器，完成停电/来电时的切换。

(4) 驱动液晶显示模块 LCD。液晶显示模块用于显示菜单、系统状态、时间、电池容量、放电指示、市电指示等。本例中采用成品的液晶模块，需要 8 个数据端口和 3 个控制端口。由 MC68HC908GP32 的 PA0～PA7 驱动数据端，PB7～PB5 驱动控制端口(图14-3中未标出)。数据端口可向液晶模块写入要显示的字符，也可以写入命令，如清除显示屏等。

(5) 检测按键输入。按键接受用户输入。其中 S1 为系统开关，S2、S3 为菜单选择按键。按下 S1 2s 后，系统启动。按下 S2、S3，用户可选择进入系统设置菜单，完成用户设置。按键状态由 MC68HC908GP32 的 PB0～PB2 口读入。在软件中判断按键状态时同样有处理抖动的问题，必须等待按键稳定后再读入状态，排除可能出现的干扰。

(6) 与 PC 机通信。串行通信采用一片 MAX232，与 MC68HC908GP32 的 SCI 口连接，与 PC 机上的通信软件通信，在 PC 机上可以查看电源的状态，并进行设置工作。由于 MC68HC908GP32 中提供了可用软件擦写的 Flash 存储器，用户设置的数据可以保存在 Flash 中，即使重新复位也不会丢失信息。

通信模块的另一个功能是：当 UPS 电源电池将耗尽时，可以向监控软件发出信号，监控软件将保存数据后关闭系统，防止电池用尽后系统掉电而丢失数据。

(7) LED 显示器和蜂鸣器。LED 显示器和蜂鸣器用于提示系统状态、错误等。本例中用引脚 PC3、PC4 驱动绿色和红色 LED 发光管，分别指示正常和错误状态；引脚 PD0 驱动蜂鸣器。

14.2.2 系统软件流程

系统软件流程如图 14-6 所示。

系统软件工作流程简述如下：

系统上电后，首先进行初始化工作，设置时钟发生器、定时器、I/O 端口、SCI 口等等。如果系统通过 RS-232 口连接了 PC 机，可以在 PC 机的监控软件上设置电源参数。如果没有，则可用键盘输入设定。需要设定的参数包括停电后的倒计时时间、电池允许放电的最大百分比等。电源启动后，将检测 220V 电网的状态，如果电网工作异常，则启动逆变器，向负载供电，同时向用户发出警告。在液晶显示器或监控软件上显示电池容量，并倒计时。如果倒计时结束或电池所剩容量低于允许值，则发出警告，而后关闭电源。

按功能划分，系统软件主要包括 4 大部分：

(1) 信号产生模块。该模块产生功率电路前后级所需的驱动信号。驱动信号的频率、占空比以及相位都要精确设定，否则将损坏强电部分电路。在软件设计中，驱动信号由定时器中断程序完成。

(2) 人机界面模块。该模块完成对液晶的初始化和系统各种信息的显示，读入用户按键输入，并执行相应操作。

(3) 通信模块。该模块完成与 PC 机的通信，传送当前 UPS 系统工作状态并接受命令设置。它主要是由 MC68HC908GP32 的 SCI 口完成。

(4) 记录模块。该模块记录用户的输入设置信息。在电源启动后还要记录电池的使

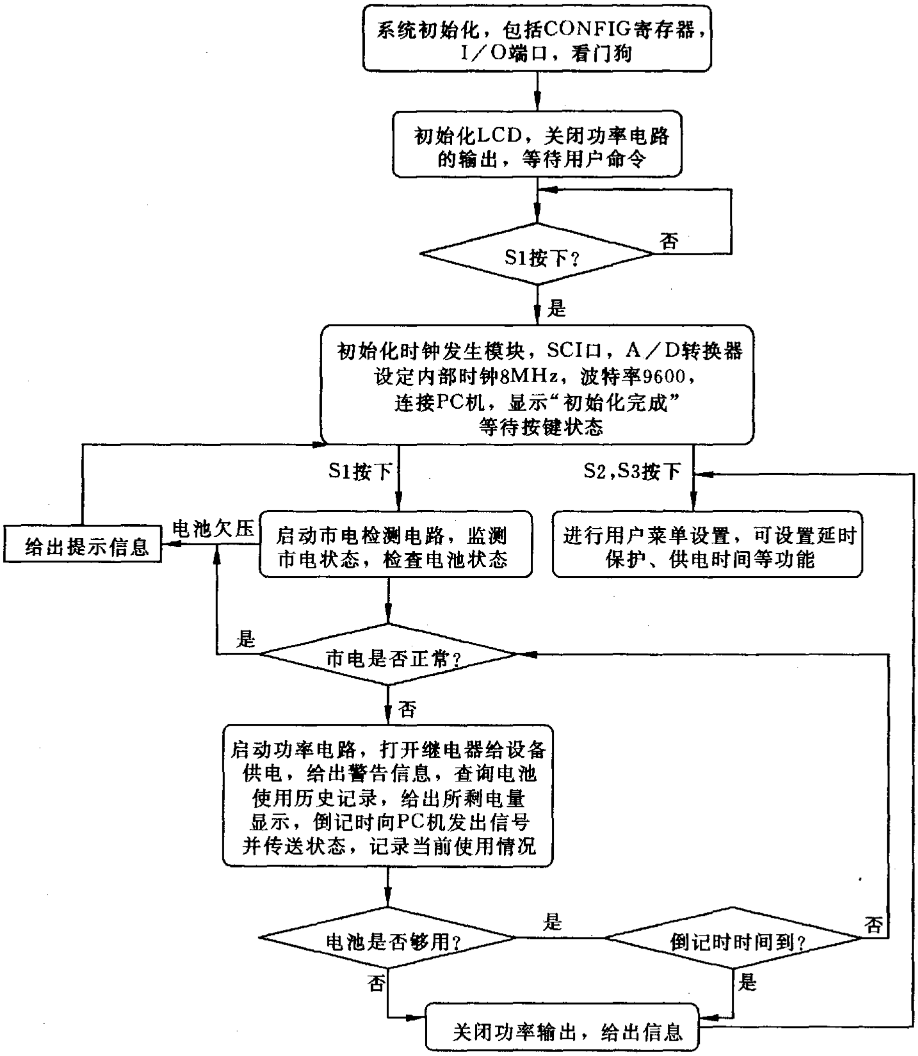


图 14-6　系统软件流程图

用时间，以便估计剩余的容量。关于 Flash 在线擦写的内容，请读者参阅 Flash 的有关章节。

每次电源启动功率电路前，系统都会检测电池状态，然后查询在 Flash 中记录的电池使用记录，计算剩余容量，并给予显示。

第15章 MC68HC908GP32 IDK 在线编程开发系统

传统的开发 Motorola 8 位微控制器的方法，是使用仿真器模拟目标系统中微控制器的运行情况，以达到调试目标系统的软件和硬件的目的。这种方法虽然便利，但不易实现100％的实时仿真。同时如果使用仿真器，开发者往往需要从最简单的模块应用入手，大大增加了开发的时间和难度。

Motorola 的微控制器进入国内较晚，大多数开发者对其性能和开发方法都不太了解。无论是芯片的销售商还是用户，都需要有一套简单易用的开发系统，可以很方便地用来对芯片的各个功能和模块进行实验和测试。

MC68HC908GP32 IDK(in system development kit)在线编程开发系统利用了微控制器内含 32kB Flash 的特点，为开发者提供了一套简便的、友好的微控制器在线编程开发系统。

一方面，用户可以简便地利用它来对芯片的各个功能和模块进行一些实验和测试。系统提供了良好的软件和硬件模块接口，用户可以直观地从目标评估母板上观察程序运行的结果，也可以从 PC 机查询微控制器当前的状态。此功能最适于高校的教学实践，是高校微控制器教学的理想实验系统。同时此功能也可以用作芯片功能的演示和测试，起到 MC68HC908GP32 评估板的作用。

另一方面，开发者可以利用我们提供的固化在微控制器中的监控程序和外围硬件功能模块，开发自己的系统。本系统在调试过程中能实现 100％的在线实时仿真；用户可以通过 PC 机方便地查询芯片状态，修改寄存器内容和用户程序。同时目标评估母板为 MC68HC908GP32 设计的各个外围硬件功能模块，可以直接嵌入到用户的各种实际系统之中，成为开发者降低开发难度、缩短开发时间的有利工具。

这套系统的原理是在 MC68HC908GP32 的 Flash 内储存一段简单、实用的监控程序，利用这段程序与 PC 机通过串行口通信，实现诸如程序下载、运行、断点设置、内存查看和修改等功能。这段监控程序控制了 MC68HC908GP32 的复位和中断服务。在芯片复位时，监控程序可以选择运行用户写入 Flash 的程序，也可以等待并执行用户通过 PC 机发出的各种调试命令，监控程序会将命令执行的结果信息通过串行口发往 PC，并显示在 PC 的屏幕上。在监控程序等待用户命令的过程中，所有的中断都是被禁止的；而在用户程序的运行过程中，一旦有中断发生，微控制器会首先执行监控程序的中断服务例程，再由这些中断服务例程去调用相应的用户程序的中断服务例程（注意，本系统并不支持用户编写的软中断(SWI)服务例程，也就是说，用户编写的软中断服务例程不会被调用）。

15.1 MC68HC908GP32 IDK 概述

15.1.1 MC68HC908GP32 IDK 的特点

MC68HC908GP32 IDK 具有如下特点：

(1) 基于 MC68HC908GP32。

(2) 内含不大于 8kB 的监控程序。

(3) 监控程序提供给用户各种基本的开发和调试功能，如程序的下载和运行、汇编和反汇编、断点设置、内存显示修改等等。

(4) 利用 MC68HC908GP32 的在线编程技术，实现了对用户程序的在线写入和对 Flash 存储内容的随时修改。

(5) 用户可以实现对自编程序的 100%在线实时仿真与监测。

(6) 目标评估母板提供了大量外围功能模块，可辅助完成对并行 I/O 口、键盘和外部中断、A/D、SPI、SCI 和时钟等模块的实验和测试，也可直接嵌入到用户应用之中。

15.1.2 MC68HC908GP32 IDK 套件

MC68HC908GP32 IDK 套件包括

(1) MC68HC908GP32 IDK 核心子板。

(2)MC68HC908GP32 IDK 目标评估母板。

(3) 6V 直流电源。

(4) RS232 串口电缆。

(5) PC 机终端仿真程序、软件编程范例、核心子板和目标评估母板原理图(3.5 英寸磁盘一张，标号为“GP32IDK”，可与清华-Motorola 单片机应用开发中心联系购买)。

注：支持 MC68HC08 汇编语言的编译器可以从 Motorola 和本中心的网站上下载，也可以直接向代理商或本中心索要。

15.1.3 MC68HC908GP32 IDK 对 PC 机的要求

MC68HC908GP32 IDK 对 PC 机的要求为

(1) 与 IBM-PC 兼容，约 640kB 的内存和 1MB 以上硬盘空间。

(2) DOS，Windows 95/98/NT/2000 操作系统。

15.2 MC68HC908GP32 IDK 的硬件及与 PC 机的连接

MC68HC908GP32 IDK 在线编程开发系统采用 RS-232 串行总线与 PC 机通信，不需

要安装任何驱动程序。用户只需依照本手册配置好核心子板和评估母板，再将其与电源和 PC 机简单地连接起来便可使用。

15.2.1 MC68HC908GP32 IDK 的硬件模块

MC68HC908GP32 IDK 硬件系统由核心子板(图 15-1)与目标评估母板(图 15-2)组成。用两对 32 芯欧式连接插头座可实现电源和信号线的连接。

核心子板提供了使 MC68HC908GP32 正常工作的最小系统，包括 MC68HC908GP32 及其复位电路、时钟电路、RS-232 串行口电路等几个部分。为了满足不同用户的需要，核心子板提供了 3 个跳线选择设置：复位电路选择(J_RST)、TxD 选择(J_TxD)和 RxD 选择(J_RxD)。

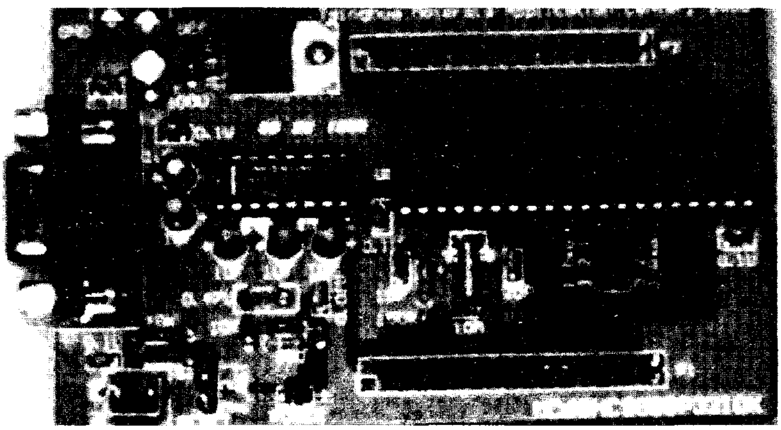


图 15-1　核心子板

目标评估母板上，除 MC68HC908GP32 IDK 模块外，还提供了多个外围硬件功能模块，包括

(1) 8 位数字量的输入输出模块。

(2) A/D 模拟电平输入模块。

(3) 液晶显示模块。

(4) 键盘和外中断信号输入模块。

(5) SPI 和 SCI 输入输出模块。

下面详细介绍各模块的功能原理。用户在了解了各模块基本的电路原理以后，使用导线在模块之间或模块内部进行连接，便可以完成各种实验。母板上提供了许多单孔插座(下文称之为"导线接插点")以方便导线的插拔。

需要特别说明的是，MC68HC908GP32 IDK 模块实际上只提供了 40 个单孔插座，其序号及其信号定义与 MC68HC908GP32 的 40 芯引脚一一对应，并通过两个欧式插座与核心子板上的 MC68HC908GP32 的同名引脚直接相连(除 RESET、TxD、RxD 外)。

为了方便初学者迅速学习和掌握各个外围硬件功能模块的使用方法，下文在介绍各个模块的同时，还提供了推荐的电路连接方法，供用户选用。

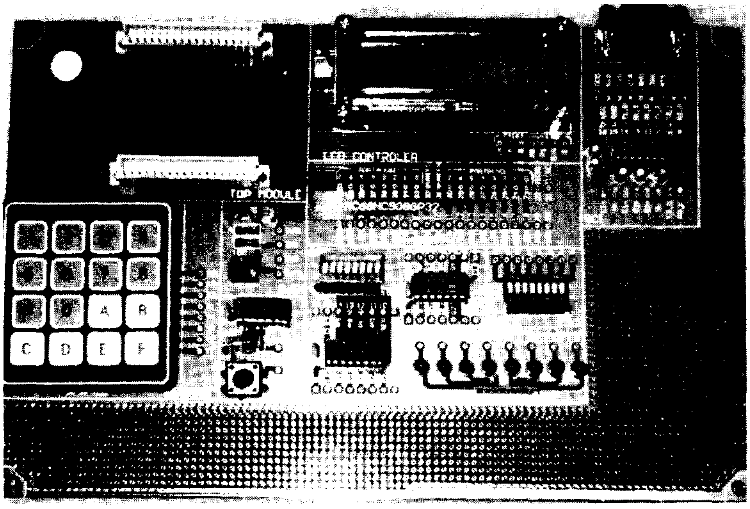


图 15-2 评估母板

在本节关于硬件电路的介绍中，提供了许多硬件原理图，其中所有在核心子板或是目标评估母板上已有的连接，都以实线表示；而所有需要用户用导线实现的连接（推荐的电路连接），都以虚线表示。

1. 核心子板与目标评估母板的接口定义

核心子板与目标评估母板之间的 2 个 34 芯欧式连接插座，用于将核心子板上的 MC68HC908GP32 的 40 芯引脚信号与目标评估板上的“MC68HC908GP32”模块的 40 芯单孔插座相连，其插座引脚信号定义如图 15-3 所示。

P1

UCC	1	2	CGMXFC
GND	3	4	OSC2
GND	5	6	OSC1
NC	7	8	RESET
GND	9	10	PC0
NC	11	12	PC1
GND	13	14	PC2
NC	15	16	PC3
GND	17	18	PC4
NC	19	20	TxD
GND	21	22	RxD
NC	23	24	IRQ
NC	25	26	SS
GND	27	28	MISO
GND	29	30	MOSI
VCC	31	32	SCK
NC	33	34	GND

P2

PA5	1	2	PA7
PA4	3	4	PA6
PA3	5	6	GND
PA2	7	8	NC
PA1	9	10	GND
PA0	11	12	NC
VREFL	13	14	GND
VREFH	15	16	NC
PB7	17	18	GND
PB6	19	20	NC
PB5	21	22	GND
PB4	23	24	NC
PB3	25	26	GND
PB2	27	28	NC
PB1	29	30	T1CH1
PB0	31	32	T1CH0
NC	33	34	GND

图 15-3 欧式连接插座

在本系统的现行版本中，只使用了以上欧式连接插座的 32 芯，即图 15-3 中 P1、P2 的 33 和 34 芯未使用。如果用户希望在核心子板和用户目标板之间通过软电缆连接，就可以选用 34 引脚的标准插座。

2. 核心子板——复位电路选择(J_RST)

用户可以通过标号为 J_RST 的 3 芯跳线来选择使用核心子板上的复位电路还是用户板(目标评估母板)上的复位电路。在本系统中，目标评估母板并没有提供任何复位电路，所以用户在使用过程中需要选择使用核心子板上的复位电路(1,2 短接，如图 15-4)。只有用户在自行开发的用户板上提供了复位电路，才可以选择使用 2,3 短接，如图 15-4 所示。

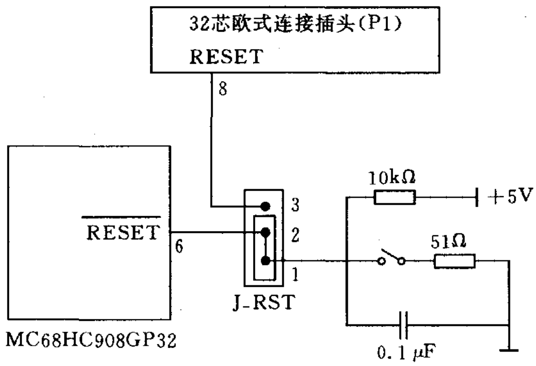


图 15-4　核心子板上的复位电路选择

3. 核心子板——TxD 选择(J_TxD)

用户可以通过标号为 J_Txd 的 3 芯跳线，来选择 MC68HC908GP32 的 SCI 模块的 TxD 数据发送线是与核心子板上的 RS-232 串行口通信电路连接，还是与用户板(或目标评估母板)上的 RS-232 串行口通信电路连接。在本系统中，MC68HC908GP32 内部驻留的监控程序需要通过核心子板上 TxD 向 PC 机发送数据，所以用户必须保证核心子板上的 TxD 与 PC 机通信线路连接的正确。在一般情况下，用户可以选择使用核心子板上的串行口通信电路(1,2 短接，如图 15-5 所示)。如果用户在目标评估母板上将 TxD 与 RS-232 通信模块进行了正确的连接，就可以选择通过母板向 PC 机发送数据(2,3 短接，如图 15-5)。

4. 核心子板——RxD 选择(J_RxD)

用户可以通过标号为 J_RxD 的 3 芯跳线，来选择将 MC68HC908GP32 的 SCI 模块的 RxD 数据接收线是与核心子板上的 RS-232 串行口通信电路连接，还是与用户板(或目标评估母板)上的 RS-232 串行口通信电路连接。在本系统中，MC68HC908GP32 内部驻留的监控程序需要通过核心子板上的 RxD 接收 PC 机发来的命令和数据，所以用户必须保证核心子板上的 RxD 与 PC 机通信线路连接的正确。在一般情况下，用户需选择核心

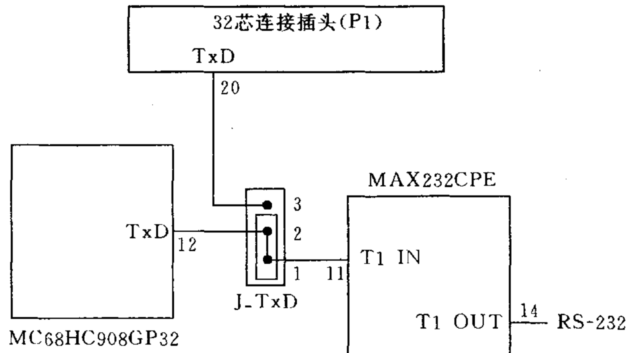


图 15-5　核心子板上 TxD 的电路选择

子板上的串行口通信电路(1,2 短接,如图 15-6)。如果用户在目标评估母板上将 RxD 与 RS-232 通信模块进行了正确的连接,就可以选择通过母板接收 PC 机发来的数据(2,3 短接,如图 15-6)。

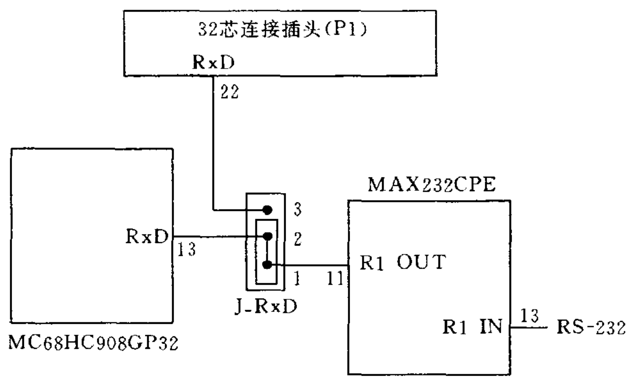


图 15-6　核心子板上的 RxD 选择电路

TxD 和 RxD 的跳线选择设置必须保持一致,即同时选择使用核心子板上的串行口通信模块,或是同时选择使用目标评估母板上的串行口通信模块。

5. 目标评估母板——8 位数字量输入

目标评估母板上的拨位开关为用户提供了 8 位数字量的输入模块,其原理图如图 15-7 所示。

当拨位开关状态为 OFF 时,相应的导线接插点上的电压为高("1");当拨位开关状态为 ON 时,相应的导线接插点上的电压为低("0")。在拨位开关两旁的"H"和"L"字母指示了开关拨动的方向所对应的电平高低。

6. 目标评估母板——8 位数字量输出

目标评估母板上的 LED 为用户提供了 8 位数字量的输出模块,其原理图如图 15-8

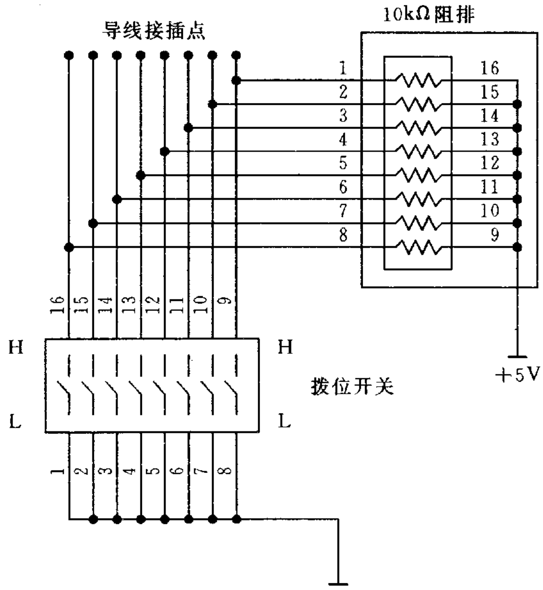


图 15-7　8 位数字量输入电路

所示。

当导线接插点的接入电平为高时,LED 处于暗状态;当导线接插点的接入电平为低时,LED 处于亮状态。

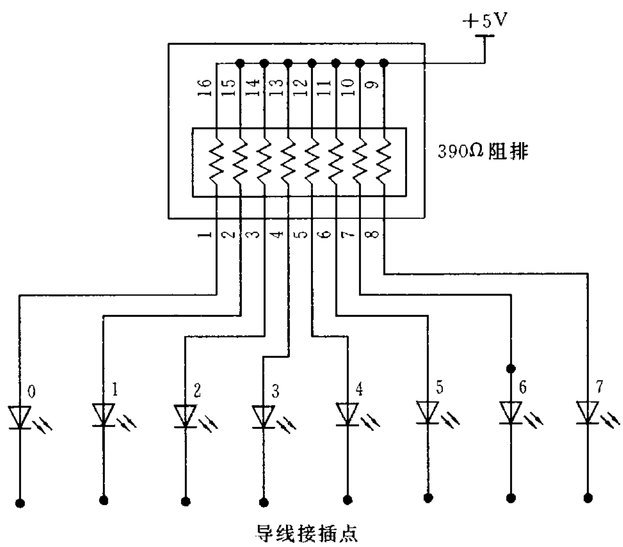


图 15-8　8 位数字量输出电路

7. 目标评估母板——A/D 模拟电平输入

目标评估母板为用户提供了两路模拟电平输入,用于 A/D 模块的模拟电平采样,其

原理图如图 15-9 所示。

其中,IN1 用于接入外电路的待测电平,经过保护电路后,电平被限制在 $-0.3V\sim5.3V$ 之间,再从 OUT1 输出。用户也可以通过调节可调电阻,在 OUT2 产生 $0\sim5V$ 的电平。

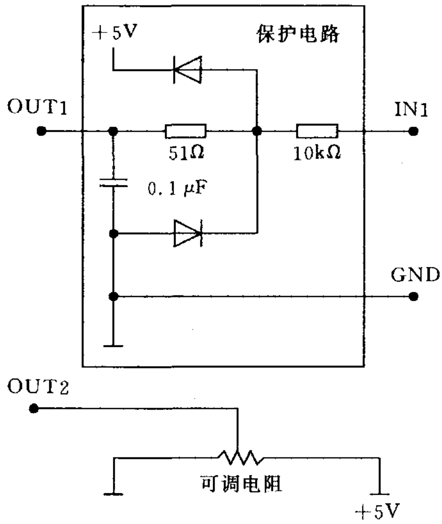


图 15-9　模拟电平输入电路

8. 目标评估母板——液晶显示

目标评估母板为用户提供了单排字符液晶,可同时显示 16 个符号或字母。

该液晶控制模块有 16 支引脚,但在本系统中仅有 14 支引脚有信号定义,分别是地线(V_{SS})、电源线(V_{DD})、驱动电压线(V_{EE})、寄存器选择信号(RS)、读/写信号(R/W)、片选信号(E)和 8 位数据线(DB0~DB7)。为了方便用户使用,我们用 74LS164 将液晶的 8 位并行数据输入方式转化为 SPI 串行输入方式。

液晶显示连接电路如图 15-10 所示。

注意:

(1) 限于篇幅,有关液晶显示模块的原理和使用方法,在本书中没有提供,用户可以参阅相关的技术文献和资料。

(2) R8_2 用于调节液晶驱动电压。改变它的阻值,可以改变液晶的显示亮度。

(3) 此功能的演示程序源代码在系统的配套软盘"GP32IDK"中可以找到。

9. 目标评估母板——键盘中断信号输入

目标评估母板为用户提供了 16 键贴膜键盘,可用于键盘中断信号的输入。键盘内部连接原理图如图 15-11 所示。

用户可以直接将键盘的 8 个导线连接点(X1~X4,Y1~Y4)与 MC68HC908GP32 的键盘中断输入口(PORTA)的 8 个引脚相连,再通过编程,就可以实现 16 键键盘输入。

此功能的演示程序源代码在系统的配套软盘"GP32IDK"中可以找到。

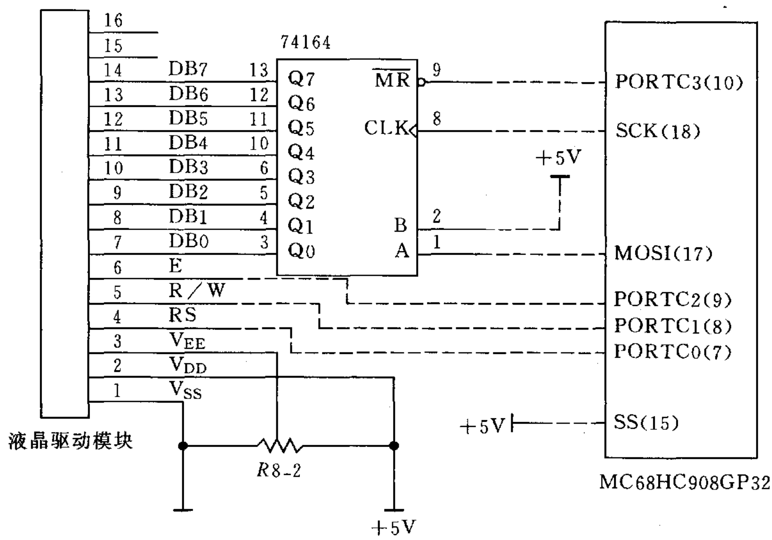


图 15-10　液晶显示连接电路

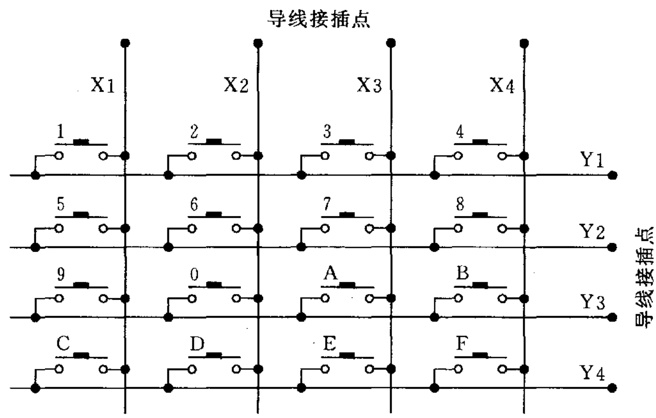


图 15-11　键盘内部连接

10. 目标评估母板——外中断信号输入

目标评估母板利用按键和 74HC14 芯片，为用户提供了正/负极性单脉冲产生模块，可用于外中断信号的输入。产生外中断信号的硬件连接原理图如图 15-12 所示。

该模块同时提供了两个导线接插点，分别对应着正极性和负极性单脉冲信号的输出。图 15-12 中，上面的接插点为正极性信号输出端，在按键未按下时保持低电平输出，按键按下时输出高电平；下面的负极性信号输出接插点，在按键未按下时保持高电平输出，按键按下时输出低电平。负极性信号可以直接接到 MC68HC908GP32 的 IRQ 引脚，作为它的外部中断信号。

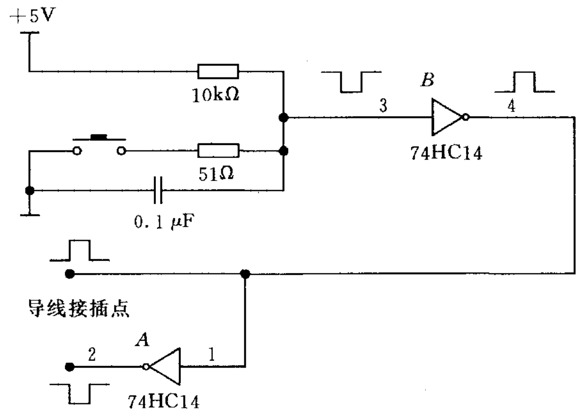


图 15-12　外中断信号产生电路

11. 目标评估母板——SPI 输入

目标评估母板为用户提供了通用芯片 74HC165，用于实现 8 位数字量的 SPI 输入。

74HC165 是 8 位并行或串行输入/串行输出移位寄存器，用户可以通过 8 位拨位开关设定 74HC165 的并行输入值，再通过编程从 SPI 读入到 MC68HC908GP32 中。

SPI 输入硬件原理和推荐连接方法如图 15-13 所示。图中，拨位开关的设置方法与 8 位数字量输入模块中的设置方法一样。74HC165 有两种数据输入方式：一是通过 P0～P7 引脚输入 8 位并行数据；一是通过 SER 引脚输入 8 位串行数据。而 PL 上的信号则决定了是从 P0～P7 输入数据还是从 SER 输入数据，所以引脚 PL 被称为串行移位/并行加载选择。当 MC68HC908GP32 进行 SPI 操作时，74HC165 中锁存的数据通过 Q7 与 MISO 的连接进入微控制器。有关 74HC165 原理和使用方法的详细说明，限于篇幅，在这里

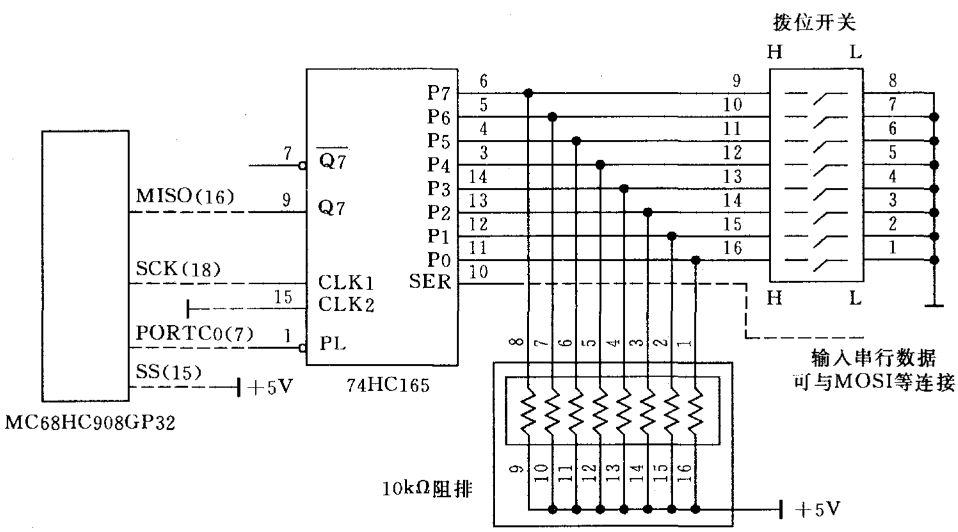


图 15-13　SPI 输入电路

不再赘述，请用户参阅相关的技术资料和文献。

12. 目标评估母板——SPI 输出

目标评估母板为用户提供了通用芯片 74HC164，用于实现 8 位数字量的 SPI 输出。

74HC164 是 8 位串行输入/并行输出移位寄存器，用户可以通过编程将 MC68HC908GP32 中储存的数据，从 SPI 输出到 74HC164 和其他输出显示设备（如 8 位数字量输出模块等）。

SPI 输出硬件原理和推荐连接方法如图 15-14 所示。

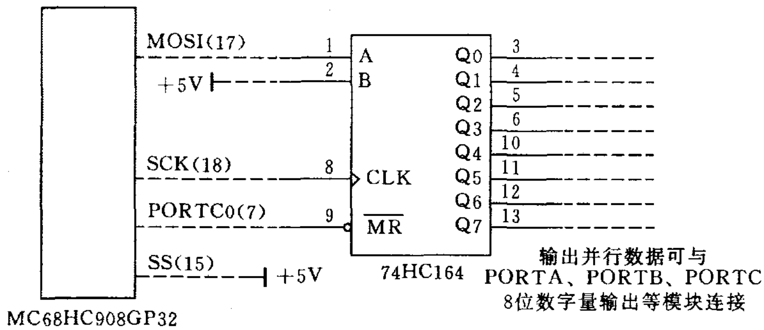


图 15-14　SPI 输出电路

74HC164 将 A、B 引脚上的信号作与运算，作为串行输入的数据信号，并移位输出到 Q0～Q7 引脚上。MR 信号用于清除 Q0～Q7 上的信号（全部输出低电平）。有关 74HC164 原理和使用方法的详细说明，限于篇幅，在这里不再赘述，请用户参阅相关的技术资料和文献。

13. 目标评估母板——SCI 输入输出

目标评估母板为用户提供了 MAX232CPE 通用接口芯片，用于实现 SCI 信号在 RS-232 总线上的输入输出。

MAX232 是 MAXIM 公司生产的，与 EIA-RS-232-E 在电气特性上兼容的串行通信收发器和驱动器，可将 MC68HC908GP32 的 SCI 信号与 RS-232 总线信号相连接，从而实

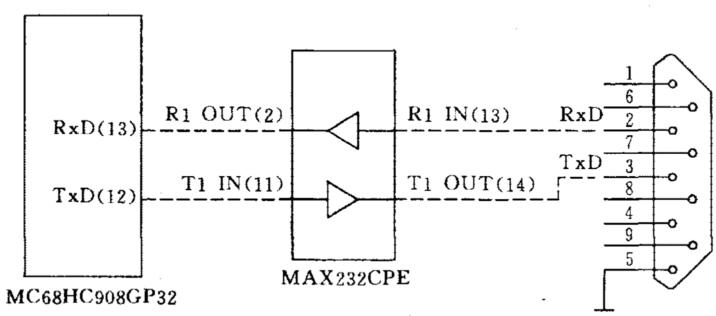


图 15-15　SCI 输入输出电路

现微控制器通过 SCI 与 PC 机的通信。

SCI 输入输出硬件原理和推荐连接方法如图 15-15 所示。

对于高级用户，该模块同样也可用于软件模拟 SCI 串行口的连接。在这种情况下，MC68HC908GP32 上用于信号输入输出的引脚不再是 SCI 模块的 TxD 和 RxD，而是时钟模块的输入捕捉和输出比较。

15.2.2 MC68HC908GP32 IDK 与 PC 机的连接

1. 电源连接

MC68HC908GP32 IDK 需要 +6V～+9V 直流电源输入。在核心子板上提供了两孔电源连接头，其中一孔标识为 GND(供地线接入)，一孔标识为 V_{CC}(供 +6V～+9V 电源线接入)。

请勿带电插拔电源导线！

2. 与 PC 机连接

MC68HC908GP32 IDK 核心子板上提供了 9 芯串行口(DP-9 MALE)，以用于与 PC 机或普通终端连接。该串行口各芯信号定义如图 15-16 所示。

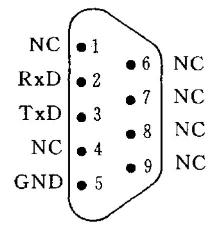


图 15-16　9 芯串行口信号定义

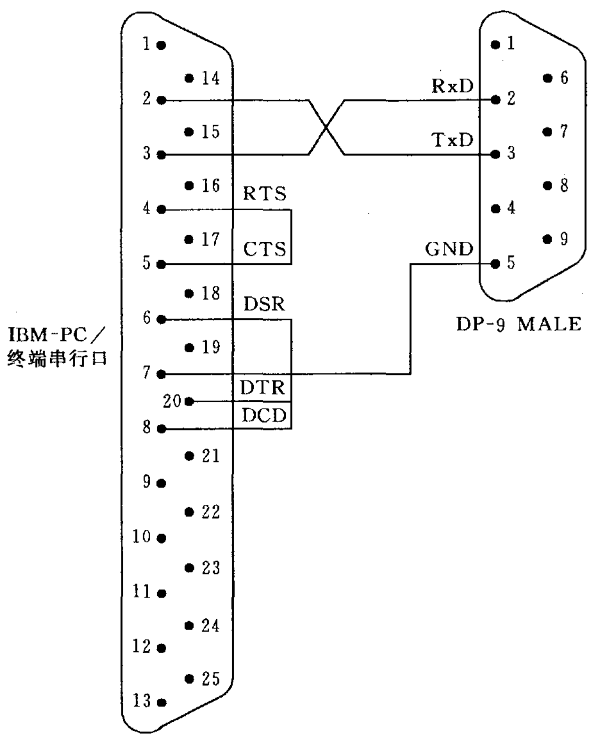


图 15-17　RS-232 串行口电缆连接

连接用电缆可以使用MC68HC908GP32 IDK套件中提供的RS-232串行口电缆，也可以自行制作。

MC68HC908GP32 IDK核心子板上的9芯串行口(DP-9 MALE)与PC机(或终端)25芯串行口的信号连接如图15-17所示。

如果用户的PC机使用的是9芯的串行口，各信号线的连接仍与图15-17一致，只是引脚号略有不同(如表15-1所示)。

表15-1　RS-232信号定义

引脚缩写符	25芯	9芯
TXD	2	3
RXD	3	2
RTS	4	7
CTS	5	8
DSR	6	6
GND	7	5
DCD	8	1
DTR	20	4

15.3　MC68HC908GP32 IDK使用指南

MC68HC908GP32 IDK在线编程开发系统依靠微控制器内部的监控程序来实现与PC机的通信，提供给用户PC机上的接口。用户可以使用普通的终端仿真软件来完成与核心子板的通信，通过向微控制器内部的监控程序传递各种命令来实现对MC68HC908GP32的在线编程和程序调试。需要指出的是，监控程序使用RS-232协议传输数据，波特率为9600 b/s，数据位为8位，停止位为1位。

本系统推荐使用PROCOMM终端仿真软件。

用户只要用RS-232电缆将核心子板和PC机正确连接，打开子板电源，监控程序就会自动初始化MC68HC908GP32，并向PC机上的终端仿真软件传递显示当前系统版本的信息。

Tsinghua Motorola MCU center IDK debug V.1.1. All right reserved,

Type 'H' for help ...... what ever you like

>

初始化完成后，监控程序向PC机传递显示监控命令提示符“>”，这时用户就可以输入各种命令了。监控程序会等待用户输入命令字母，并提示相应的参数输入。当一个正确的命令输入后，监控程序会进入下述两种基本的工作方式：

(1) 当用户输入某监控命令时，监控程序执行该监控命令相应的服务例程，并在例程执行结束后在终端屏幕上显示“>”提示符，此时用户就可以输入新的监控命令了，其中有

些监控服务例程在执行过程中，还会要求用户输入数据。

(2) 当用户要求运行一个用户程序时，若用户设置了断点，则监控程序从当前 PC 处开始执行用户程序，运行到断点处返回监控状态，终端屏幕上显示提示符“>”；而若用户未设置断点，则监控程序就将微控制器控制权完全交给用户程序，即在微控制器复位之前，用户将无法输入任何监控命令。

如果用户已经向 MC68HC908GP32 下载了自行编写的用户程序，并在用户中断向量表(见 15.4.1 节说明)中填写了复位向量，系统复位时会自动等待用户选择进入监控程序还是用户程序。如果用户在复位后 5s 内没有任何键盘动作，系统会自动从用户程序的复位向量处开始运行。

15.3.1 监控命令列表

监控程序 1.1 版本中支持的监控命令如表 15-2 所示。

表 15-2 监控命令表

命令格式	操　作	备　注
A xxxx	反汇编和汇编 (起始地址为 xxxx)	用户可以直接输入汇编源码(或机器码)以改变 Flash 中的语句内容
B xxxx	设定断点(地址为 xxxx)	
Ctrl + A	改变累加器(A)	
Ctrl + X	改变变址寄存器(H:X)	
Ctrl + C	改变条件码寄存器(CCR)	
Ctrl + P	改变程序计数器(PC)	
D xxxx－xxxx	显示指定地址内的存储内容	对 RAM、Flash 和寄存器都有效
E xxxx－xxxx	擦除指定地址内的存储内容	仅对 Flash 有效
G	运行用户程序	入口由 PC 寄存器指定
H	显示所有命令的帮助信息	
L	下载 S 记录格式文件到 Flash 中	
M xxxx	改变指定地址的存储内容	同时支持对 RAM 和 Flash 的修改
R	显示微控制器寄存器	包括 A、H、X、CCR、PC、SP 等

只要用户输入了完整的命令行(包括命令字母和参数)，监控程序就会立即执行相应操作，不需要用户以回车来结束命令行的输入。

在命令输入过程中，如果参数还未输完就敲击回车或是输入不正确的参数，该命令不会被响应。

15.3.2 监控命令描述

1. A——反汇编和汇编命令

A 命令用于执行反汇编和汇编服务例程。在键入命令时，用户需要同时输入目标代码的起始地址（4 位 16 进制）。例程会显示从起始地址反汇编出的第一条指令，并以“<<.”符号提示用户输入新的指令。当用户输入新的汇编指令，并以回车结束时，监控例程会汇编出新指令的机器码，并自动写入 Flash（如果用户在输入过程中使用退格键，监控例程会忽略用户的输入，重新显示当前指令，并提示用户输入）；当用户直接键入回车时，监控例程会继续显示下一条指令以供查看或修改；当用户键入“.”时，终止此次汇编/反汇编命令。

例 15.1

```
>A 8000
@8000 45 02 3F   LDHX # $023F <<. LDHX <Backspace>
@8000 45 02 3F   LDHX # $023F <<. LDHX # $0000<CR>
@8000 45 00 00
@8003 94         TXS         <<. <CR>
@8004 A6 01      LDA # $01 <<. .
>
```

2. B——断点设置命令

B 命令用于设置程序运行断点。在键入命令时，用户需要同时输入断点地址（4 位 16 进制）。如果用户不输入任何地址，而直接以回车结束命令，例程会清除当前的断点设置，禁止断点功能。需要注意的是，MC68HC908GP32 只允许同时设置一个断点，所以如果用户重复设置断点，则只有最后设置的断点有效。一旦用户程序运行到断点地址，监控程序就会暂时停止正在运行的用户程序，显示当前寄存器状态，提示用户执行新的命令或是继续运行程序，同时清除当前的断点设置，用户需要重新使用 B 命令设置断点。

例 15.2

```
    设置断点：
>B 8003
>
    程序运行到断点处：
BreakPoint Encountered, Register Values :
A=04 H:X=023F PC=8003 CC=68 SP=023A
>
```

3. Ctrl+A/X/C/P——修改 A/(H:X)/CCR/PC 内容命令

该命令用于修改累加器 A、变址寄存器（H:X）、条件码寄存器（CCR）和程序计数器

(PC)的内容。用户键入命令后，服务例程会显示指定寄存器的当前值，并以“.”提示用户输入。用户可以输入新值，也可以直接键入回车结束。

例 15.3

```
>^A     A=04.05
>^X     H:X=023F.0000
>^C     CC=68.<CR>
>^P     PC=8003.8000
>
```

4. D——显示存储内容命令

该命令用于显示 MC68HC908GP32 的存储区内容，包括 RAM($0040～$023F)和 Flash($8000～$DFFF)等。在键入命令时，用户需要同时输入要显示的存储区的起始地址(4 位 16 进制数)和结束地址(4 位 16 进制数)。服务例程会同时显示存储内容的 16 进制和 ASCII 值(如果是不可显示的 ASCII，则以“.”代替)。如果用户输入的地址范围内存在着非存储单元，则显示的数据可能是 $81 或是其他任意值。

例 15.4

```
>D E070-E080
E070 BD AF 01 65 00 10 26 F4 CD E3 7C 6E FF 04 45 00 ...e..&...|n..E.
E080 00 D6 E0 8F AF 01 A1 04 27 59 CD F3 77 20 F2 0D ........'Y..w..
>
```

5. E——擦除 Flash 存储内容命令

该命令用于擦除 MC68HC908GP32 的 Flash 存储区内容。在键入命令时，用户需要同时输入要擦除的存储区的起始地址(4 位 16 进制数)和结束地址(4 位 16 进制数)。由于监控程序驻留的区域($E000～$FDFF)不允许擦除，所以该命令只对部分 Flash 存储区($8000～$DFFF)有效。在擦除操作完成后，服务例程会对该 Flash 存储区进行空校验，并显示提示信息表示是否擦除成功。

例 15.5

```
>E 8000-D000
8000-D000 Successfully erased.
>E 8000-F000
8000-F000 Error erased.
>
```

6. G——程序运行命令

该命令使 MC68HC908GP32 从当前程序计数器(PC)所指向的代码地址开始运行，

所以常需要和 Ctrl+P 命令配合使用，用于运行用户下载到 Flash 中的程序。该命令没有任何命令参数。如果用户没有设置断点，则在下一次 MC68HC908GP32 复位以前，监控程序将无法接收任何用户发出的命令，MC68HC908GP32 将完全被用户程序所占用。MC68HC908GP32 复位以后，监控程序会重新控制 MC68HC908GP32，等待用户输入新的命令。

例 15.6

```
>^P PC=0000.8000
>G
```

7. H——显示命令列表和帮助信息

该命令显示监控程序提供的所有命令及帮助信息，没有任何命令参数。

例 15.7

```
>H
^A,^X,^C,^P : Register Change          A xxxx : Assembler/Disassembler
D xxxx-xxxx : Memory Display           M xxxx : Memory Modify
G : Run Program                        B xxxx : Set Break Point
L : Download S19 File To Flash         R : Register Display
E xxxx-xxxx : Erase Flash              H : Display This Message
>
```

8. L——下载程序命令

该命令从 PC 机下载 S 记录格式文件到 MC68HC908GP32 的 Flash 中。用户在键入"L"命令后，可以使用终端仿真程序下载编译好的 S 记录格式文件。关于使用 PROCOMM 软件下载文件的方法，请参看本书第 15.6 节中 PROCOMM 的使用说明。在文件下载过程中，监控程序会实时显示接收到的内容，并在下载结束后显示操作是否成功。需要注意的是，用户编写的源程序必须满足本系统的要求，详见本书第 15.4 节。而且在用户程序编译生成 S 记录格式文件后，并不能直接用于下载，需要使用本系统提供的 CHS19 程序进行格式调整，详见本书第 15.5 节。用户也可以直接利用 WIN-IDE 软件对程序进行编译，生成 S19 记录格式文件，并可将其直接用于下载，详见第 15.7.1 节。

例 15.8

```
>L
S12380009BCC81860000804A00000000013F0000801A00008022801A801C0000FFFF000074
S1238020000000009BA601B71EA63DB71F3F366E01366E04386EB0396EFF3A1A36163614FA
S90380007C
    Successfully writing Flash memory.
>
```

9. M——修改存储区内容命令

该命令用于修改 MC68HC908GP32 的存储区内容，包括寄存器区（\$0000～\$003F）、RAM（\$0040～\$023F）和 Flash（\$8000～\$DFFF）等。在键入命令时，用户需要同时输入要修改的存储区字节的地址（4 位 16 进制数）。服务例程会自动显示该字节存储的 16 进制值，并用“.”提示用户输入。用户可以输入新的 16 进制值，例程会自动将新值写入，并显示该字节最终的存储内容，同时显示下一字节内容，以供查看或修改（如果用户在输入过程中使用退格键，例程会忽略用户的输入，重新显示当前字节，并提示用户输入）；用户也可以直接键入回车，例程会略过当前字节而继续显示下一字节内容；或是键入“.”，终止此次修改命令。如果用户输入的地址不在可修改的存储区内，则服务例程提示用户输入有误。

例 15.9

```
>M F000
          Address Invalid (Not in RAM or Flash)
>M 8000
8000      9B.1<Backspace>
8000      9B.11
8000      11
8001      CC.<CR>
8002      81..
>
```

10. R——显示各寄存器内容

该命令用于显示累加器 A、变址寄存器（H:X）、条件码寄存器（CCR）、程序计数器（PC）和堆栈寄存器（SP）的内容。

例 15.10

```
>R
      A=04  H:X=023F  PC=8004  CC=68  SP=023A
>
```

15.3.3 MC68HC908GP32 IDK 使用注意事项

下面介绍使用时可能会遇到的一些问题。

(1) 核心子板与 PC 机处于带电连接状态时，请不要随意插拔 RS-232 串行电缆。如 RS-232 电缆连接出现不正常的情况，请关掉 PC 机电源，在正确连接电缆后重新启动系统。

(2) 如果核心子板与 PC 机正常连接后用户仍然无法通过终端仿真软件与核心子板建立通信，请关闭终端仿真程序，重新确认核心子板的电源连接并复位，然后重新运行终

端仿真程序。

15.4 MC68HC908GP32 IDK 的软件编程

MC68HC908GP32 IDK 在线编程开发系统为用户提供了功能强大的在线编程调试环境，用户可以将自己编写的代码在编译后直接下载到 MC68HC908GP32 的 Flash 中，进行 100%的实时在线调试。为了发挥监控程序的最大作用，用户需要按本系统的要求进行编程，包括程序代码和变量的地址分配、中断向量表的填写等等。

15.4.1 程序代码和变量地址分配

在本系统中，MC68HC908GP32 的部分 RAM 和 Flash 被监控程序所占用，为了避免与用户程序冲突，系统重新定义了 MC68HC908GP32 的寻址空间，如图 15-18 所示。

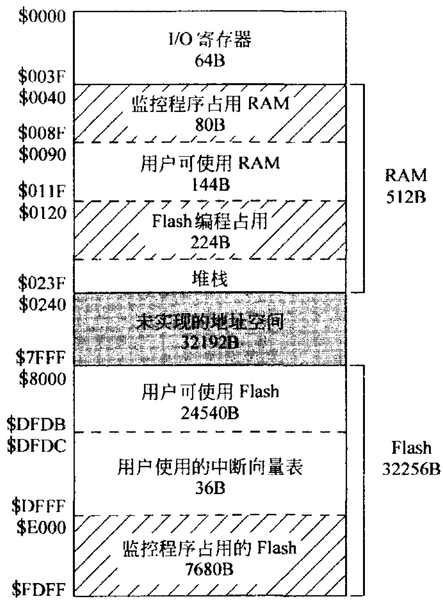


图 15-18 MC68HC908GP32 IDK 寻址空间分配

MC68HC908GP32 IDK 用户中断向量如表 15-3 所示。

用户在编写用户程序时，需要将变量放在 RAM 的 $0090～$011F 区域内，程序代码放在 Flash 的 $8000～$DFDB 区域内，中断向量填写在 Flash 的 $DFDC～$DFFF 的中断向量表内。

15.4.2 程序初始化

除了 MC68HC908GP32 的内存分配以外，用户还要注意的就是监控程序对

MC68HC908GP32 做的初始化操作。在用户程序开始运行之前，监控程序已经对部分寄存器进行了设置。用户需要了解这些设置，以免在程序调试过程中出现意想不到的情况。

表 15-3　MC68HC908GP32 IDK 用户中断向量表

优先级	向量名	地址	描　述
最低	IF16	$DFDC	时基(高)
↓		$DFDD	时基(低)
	IF15	$DFDE	ADC 转换结束(高)
		$DFDF	ADC 转换结束(低)
	IF14	$DFE0	键盘(高)
		$DFE1	键盘(低)
	IF13	$DFE2	SCI 发送(高)
		$DFE3	SCI 发送(低)
	IF12	$DFE4	SCI 接收(高)
		$DFE5	SCI 接收(低)
	IF11	$DFE6	SCI 出错(高)
		$DFE7	SCI 出错(低)
	IF10	$DFE8	SPI 发送(高)
		$DFE9	SPI 发送(低)
	IF9	$DFEA	SPI 接收(高)
		$DFEB	SPI 接收(低)
	IF8	$DFEC	时钟 2 溢出(高)
		$DFED	时钟 2 溢出(低)
	IF7	$DFEE	时钟 2 通道 1(高)
		$DFEF	时钟 2 通道 1(低)
	IF6	$DFF0	时钟 2 通道 0(高)
		$DFF1	时钟 2 通道 0(低)
	IF5	$DFF2	时钟 1 溢出(高)
		$DFF3	时钟 1 溢出(低)
	IF4	$DFF4	时钟 1 通道 1(高)
		$DFF5	时钟 1 通道 1(低)
	IF3	$DFF6	时钟 1 通道 0(高)
		$DFF7	时钟 1 通道 0(低)
	IF2	$DFF8	锁相环 PLL(高)
		$DFF9	锁相环 PLL(低)
	IF1	$DFFA	外中断 IRQ(高)
		$DFFB	外中断 IRQ(低)
	—	$DFFC	软中断 SWI(高)
		$DFFD	软中断 SWI(低)
	—	$DFFE	复位(高)
最高		$DFFF	复位(低)

监控程序初始化时对寄存器的设置如表 15-4 所示。

表 15-4 监控程序初始化时对寄存器的设置值

地址	寄存器名	设置值(十六进制)	描述
$001E	CONFIG1	01	属性设置
$001F	CONFIG2	3D	
$0036	PCTL	71	锁相环(产生 2.4576MHz 内部总线时钟)
$0037	PBWC	E0	
$0038	PMSH	01	
$0039	PMSL	2C	
$003A	PMRS	80	
$0013	SCCR1	40	SCI(允许 9600bps 双向传输速率)
$0014	SCCR2	0C	
$0019	BAUD	02	
$FE00	SBSR	10	断点中断
$FE03	SBFCR	80	
$FE0B	BRKSCR	00	

表中未列出的寄存器在监控程序初始化时没有使用,仍保持 MCU 复位后的状态。

15.4.3 MC68HC908GP32 IDK 中的用户程序与目标系统中用户程序的关系

如果用户希望自己的程序能脱离监控程序的控制而自由运行,可以依照本节介绍的两个步骤,逐步实现用户程序在目标系统的独立运行。

第一,使用户复位向量($DFFE~$DFFF)为非空(不为$FFFF),这时监控程序会在 MCU 复位后等待用户从 PC 机发来的键盘命令。如果用户在 5s 内没有任何键盘动作,监控程序会放弃对 MCU 的控制,跳转到用户程序的复位向量处,开始运行用户程序。在这种情况下,监控程序与用户程序并存于 MC68HC908GP32 芯片中,监控程序做完本书第 15.4.2 节所述的初始化工作后,就会将 MC68HC908GP32 芯片控制权完全交给用户程序。

如果用户希望在用户程序的运行过程中能返回监控程序,就可以使用跳转语句(JMP)返回到监控程序的入口地址。该入口地址储存在$DFDA~$DFDB。

第二,当用户程序经过第一步,被确认程序功能完全正常以后,用户可以按照如下过程将用户程序写入一片空的 MC68HC908GP32 中。实现这一步,需要做以下工作:

(1) 将监控程序所做的初始化代码添加在用户程序入口处。用户可以在系统配套软盘“GP32IDK”中找到这些代码(initial.asm)。对于熟练的开发人员,照搬监控程序的初始化代码也许并不是最佳选择,用户完全可以参照本书第 15.4.2 节的介绍,并根据自己的实际需要来编写初始化代码。

(2) 修改用户向量表,即用户需要将自己的中断服务程序入口地址填写到 MC68HC908GP32 指定的中断向量表中去(MC68HC908GP32 的中断向量表可以在本书附录或芯片手册中查到)。实现的方法很简单。在本系统中,用户实际上只需要将用户向

量表的地址加上＄2000，便可以得到芯片的向量表地址。例如，用户复位向量在＄DFFE～＄DFFF，而芯片的复位向量在＄FFFE～＄FFFF。

完成上述步骤，用户程序就可以完全脱离监控程序而存在了。此时，用户可以使用专用编程器将修改后的程序写入到一片新的MC68HC908GP32芯片中，再将芯片插入目标系统，用户程序就可以脱离监控程序而自由运行了。

15.5 用户程序的编辑、编译及调试

本节通过一个范例说明用户程序的编辑、编译及调试的过程。

本范例实现PORTA输出8位二进制数值的循环移位。如果将PORTA各引脚与目标评估母板上的LED一一连接，就会产生走马灯的效果。同时程序也支持外部中断，一旦外部中断发生，PORTA的输出数值将被复位为初始值。下面详细介绍从源代码编写到程序下载、调试的每一步骤。

15.5.1 编辑源代码

下面是源代码(可在“GP32IDK”软盘中找到)。

```
** ======  寄存器  ====          ;GP32寄存器定义
PORTA     EQU     $0000          ;PORTA数据寄存器
DDRA      EQU     $0004          ;PORTA方向寄存器
INTSCR    EQU     $001D          ;中断状态控制寄存器
*
BEGIN     EQU     $8000          ;程序起始地址，全部代码均位于
*                                ;用户可使用的Flash内
** ==========主程序
*
   ORG BEGIN
INIT:
*
* ==========初始化=========
*
     LDA   #$05                  ;允许外部中断
     STA   INTSCR
     MOV #$FE,PORTA              ;初始化PORTA数值为$FE
     MOV #$FF,DDRA               ;PORTA方向为输出
     CLI                         ;清除中断屏蔽位
LOOP:                            ;开始循环移位
     LDHX #$8000
```

```
LOOP1:                                  ;H:X 逐次递减直至为 0
     AIX    #1
     CPHX # $0000
     BNE    LOOP1
     LDA    PORTA                        ;PORTA 数值移位
     SEC                                 ;最高位移出,最低位置 1
     ROLA
     CMP    # $FF
     BNE NEXT                            ;如果 PORTA 为 $FF
     LDA    # $FE                        ;则恢复 PORTA 为 $FE
NEXT:
     STA    PORTA
     BRA    LOOP
*
* === ISR_IRQ ===                        ;外中断服务例程
*
ISR_IRQ:
     LDA    # $05                        ;清除中断标志位
     STA    INTSCR                       ;允许下一次中断
     MOV    # $FE,PORTA                  ;复位 PORTA 为 $FE
     RTI
*
* === 向量表 ===                          ;填写用户向量表
*
       ORG     $DFFA                     ;用户向量表中的外中断向量
IRQ_VECTOR DW ISR_IRQ
```

在编写过程中,用户需要注意:

(1) 监控程序已为用户完成芯片的初始化和部分设置(参看本书第 15.4.2 节),用户只需设置程序流程中需要用到的寄存器即可;

(2) 中断向量必须填写在用户向量表中指定的位置(参看本书第 15.4.1 节)。

15.5.2 编译源程序

用户可以使用任何支持 MC68HC08 汇编语言的编译器编译自己的程序。Motorola 公司和清华 Motorola 单片机应用开发研究中心的网站都提供了这类编译器,用户可下载。

15.5.3 将编译好的 S 记录格式文件调整为适合下载的格式

用户需要使用 MC68HC908GP32 IDK 套件软盘中的 CHS19. EXE 程序调整 S 记录

格式文件的格式。该程序的命令行如下：

(drive):\(path)...\ CHS19. EXE <filename>

filename 为已经编译好的 S 记录格式文件名(不可省略后缀名)。CHS19 会自动将新生成的文件命名为 RESULT. S19,用于从 PC 机下载到 MC68HC908GP32 中。

15.5.4 与核心子板建立通信

用户需要正确连接核心子板和 PC 机,开启电源,并启动终端仿真程序进行设置和连接,直至出现命令提示符">"(使用终端仿真程序 PROCOMM 与核心子板建立通信的方法请参看本书第 15.6 节)。

15.5.5 下载程序

输入"L"命令,监控程序开始等待文件的传送。使用终端仿真软件的文件传输命令,将 RESULT. S19 文件发往核心子板。用户程序下载完成后,监控程序会显示提示信息,并等待用户输入新的命令。注意在下载之前,用户需要确认程序下载的目标 Flash 区域为已擦除状态(所有字节为 $FF),否则在程序下载过程中监控程序可能会提示出错。这时用户可以先擦除目标 Flash 区域中的所有内容,再重新进行程序下载操作。

15.5.6 设定程序运行起点并开始调试

使用"Ctrl+P"命令设置程序计数器(PC)为 INIT 标号指向的地址(即 $8000),然后用"G"命令使微控制器从 INIT 标号处开始运行。用户也可以在运行程序前设置断点,以对程序进行调试。

15.6 终端仿真程序 PROCOMM 的使用说明

PROCOMM 是一个功能丰富的终端仿真程序,可以直接控制调制解调器以完成诸如拨号、应答、建立通信链路等多种通信功能。但在本系统中,用户只需要使用 PROCOMM 提供的友好的终端仿真界面来与核心子板进行 RS232 串行口通信,并不需要太多的协议处理和状态控制。所以下文只提及用户在使用本系统时需要掌握的命令和设置。如果用户希望了解 PROCOMM 的其他功能,可以参阅 PROCOMM 的联机帮助。

注:Windows 自带的超级终端也是一种终端仿真程序,使用方法详见第 15.7.2 节。

15.6.1 PROCOMM 简介

用户在 DOS 或 WINDOWS 环境中直接运行 PROCOMM. EXE 应用程序,可以启动

PROCOMM，出现的用户界面如图 15-19 所示。

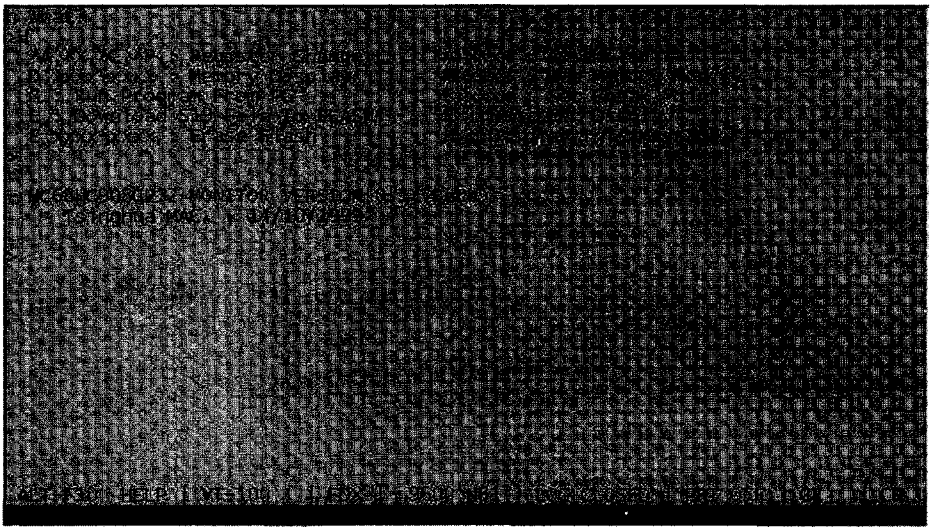


图 15-19　PROCOMM 用户界面

PROCOMM 的用户界面分为用户区和状态条两部分。用户区用于显示从 PC 机串行口读入的数据(MC68HC908GP32 IDK 核心子板发出)，状态条则显示 PROCOMM 的部分当前设置和帮助信息。进入用户界面后，用户可以通过键盘输入命令和数据。PROCOMM 会根据用户的按键判断是否是 PROCOMM 预定义的命令键(例如 Alt＋Z，用于设置 PROCOMM 的显示颜色)，如果是，PROCOMM 会进行相应的操作，否则将用户键入的数据原样发送到 PC 机的串行口(发往 MC68HC908GP32 IDK 核心子板)。

用户可以同时按下 Alt 和 F10，以显示 PROCOMM 的命令帮助信息，如图 15-20 所示。

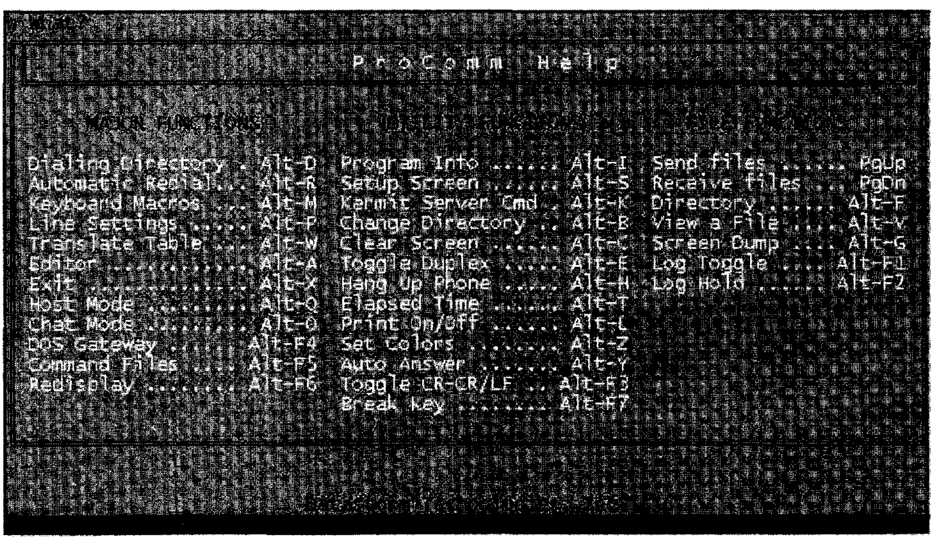


图 15-20　PROCOMM 帮助信息

其中用户在使用本系统时需要掌握的命令如表 15-5。

表 15-5　PROCOMM 部分命令列表

快捷键	命令名称	功能描述
Alt+C	Clear Screen	清屏
Alt+F10	Help	帮助信息
Alt+P	Line Setting	设置 RS-232 串行口
Alt+X	Exit	退出 PROCOMM
PgUp	Send Files	发送文件

15.6.2　PROCOMM 设置

在使用 PROCOMM 与 MC68HC908GP32 IDK 核心子板通信以前，需要对 PROCOMM 进行设置，包括通信协议、传输速率等等。在一般情况下，PROCOMM 的大多数属性都不需要用户重新进行设置，用户只需使用 Line Setting 命令设置相关的 RS-232 串行口属性即可。

如图 15-21 所示，用户可以通过 Line Setting 命令设置 RS-232 串行口的标号、通信协议和速率等。在本系统中，用户需要选择与 MC68HC908GP32 IDK 核心子板相连接的串行口(如 COM1)，设定波特率为 9600b/s，无奇偶校验，数据位为 8 位，停止位为 1 位(即 9600,N,8,1)。保存设置后返回用户界面。

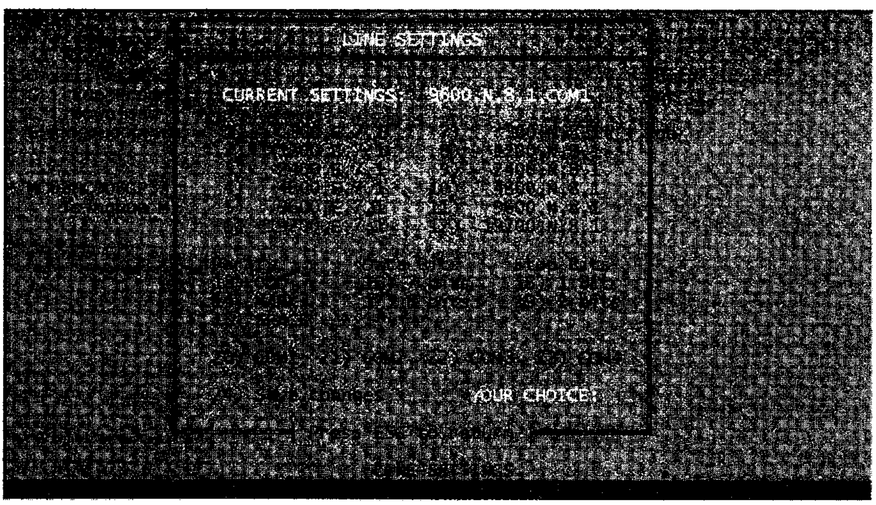


图 15-21　PROCOMM 串行口设置

15.6.3　PROCOMM 通信

PROCOMM 设置完成后，从 MC68HC908GP32 IDK 发送的数据会一一显示在用户

区中。用户可以输入各种监控命令(参看 15.3.3 节中的命令描述),包括程序下载命令“L”。

在输入“L”命令后,监控程序会等待 S 记录格式文件的发送。用户可以使用 PROCOMM 的 Send Files 命令发送指定的文件。用户首先需要按下快捷键 PgUp,这时 PROCOMM 会显示 UPLOAD 对话框,如图 15-22 所示。

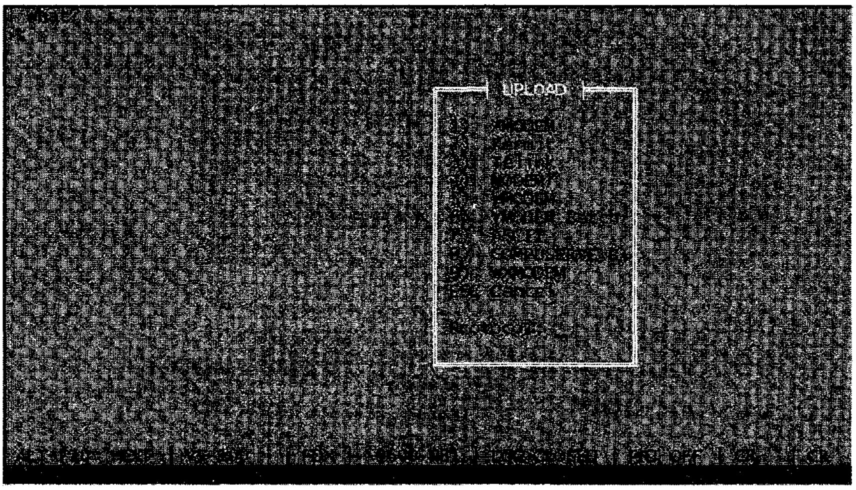


图 15-22　UPLOAD 对话框

用户需要选择第 7 项协议,即发送 ASCII 文件(键入“7”)。PROCOMM 会提示用户输入 ASCII 文件名,如图 15-23 所示。

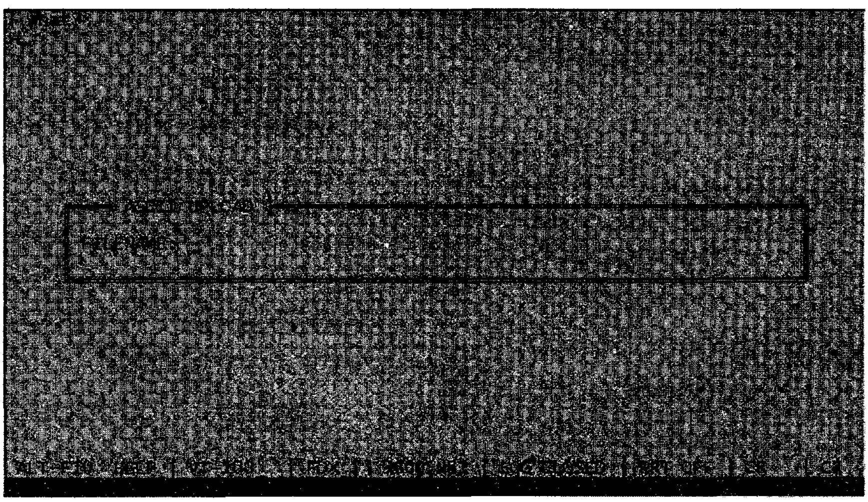


图 15-23　文件名输入对话框

用户需要输入要下载的 S 记录格式文件的全路径名(如 C:\RESULT.S19)。PROCOMM 会自动将指定的 S 记录格式文件发往设定的串行口,并回到用户界面。待监控程序将程序下载完成后,用户便可输入新的命令进行用户程序的调试。

15.7 MC68HC908GP32 IDK 集成开发环境使用说明

15.7.1 8 位 Motorola 微控制器汇编程序编译环境

同一般的微控制器或是微处理器的编译原理一样，Motorola 微控制器汇编语言可以通过任何支持 CPU08 指令系统的编译器编译用户程序，将源代码转换成微控制器可以识别并执行的机器语言代码。

P&E 微计算机系统公司的编译软件是 WIN－IDE。下面简要介绍 WIN－IDE(1.10 版)的常用命令和设置。其他功能，可参阅 WIN－IDE 的联机帮助。

1. WIN－IDE 的用户界面

若计算机上已经安装了 WIN－IDE 软件，读者执行如下步骤：[开始]→[程序]→[ICS08GP Development Kit]→[ WinIDE Development Environment]，就可以进入到 WIN－IDE 中，可以看到如图 15-24 所示的程序界面。

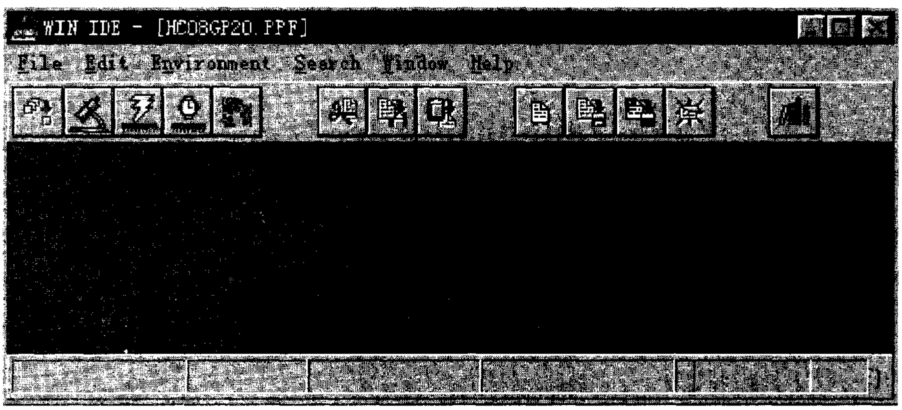


图 15-24 WIN-IDE 程序界面

与通常的软件一样，WIN－IDE 的界面也分为菜单栏、图标栏和编译区 3 个部分。

2. 编辑源程序

在图 15-24 所示界面中，点击“File”中的“New File”，弹出一个白色区域窗口，如图 15-25 所示。读者可以在白色区域中编辑自己的源程序。

3. 编译源程序生成 *.S19 文件

编辑好源程序后，读者可以将其存盘，这时生成 *.asm 文件。然后，按下图标栏中的 ，则系统执行 Assemble/Compile File 功能，组合和编译文件。如果用户程序有错，编译将停止，返回源程序，并提示错误地点和类型，并等待修改。当读者的程序正确，并通过

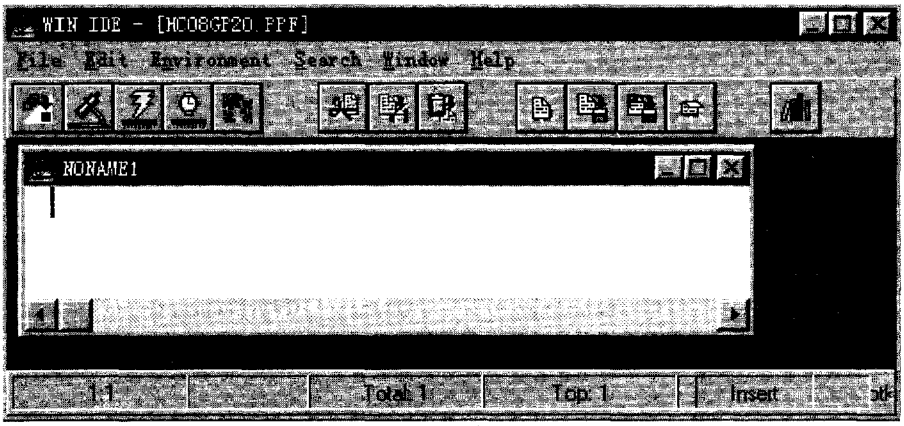


图 15-25 源程序编辑对话框

了编译后,系统将生成 *.S19 文件,并自动地将其存于 *.asm 源文件所在的文件夹中。*.S19 文件可下载到 HC908GP32 芯片中。

15.7.2 终端仿真程序——超级终端

使用 Windows 自带的"超级终端",可以建立 PC 与 MC68HC908GP32 芯片的通信。读者可以在"开始"菜单中找到该程序。步骤是:[开始]→[程序]→[附件]→[通信]→[超级终端]→[Hypertrm]。这时,程序启动,出现的图象如图 15-26 所示。然后,出现用户界面,如图 15-27 所示。键入用户名,则建立了与芯片的一个通信连接(注意:此时应保证 IDK 与 PC 已通过串行口线连接在一起)。按"确定"后,出现如图 15-28 所示的对话框,进行串行口通信设置,将"连接时使用(N):"设置为"直接连接到串口"。按"确定",弹



图 15-26 超级终端启动界面

出如图 15-29 所示对话框，进行串行口设置。各参量设置如图 15-29 所示。最后，按“确定”。当出现如图 15-30 所示的窗口时，表明 PC 已和芯片建立了通信。然后，就可以按照第 15.7 节中所述的操作方法，实现各种功能。



图 15-27　新建连接

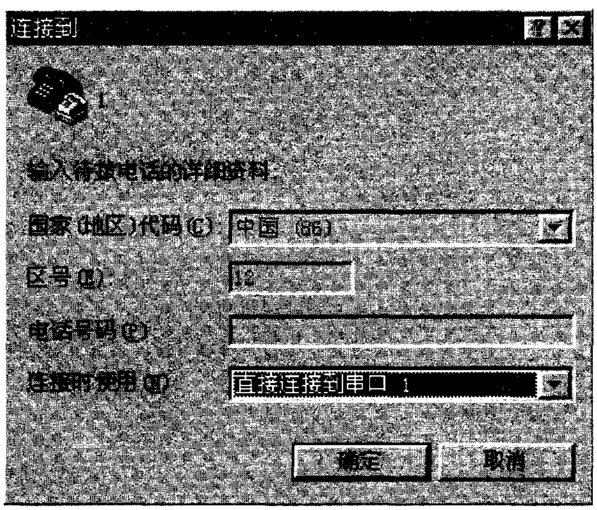


图 15-28　设定通信方式

注意：这里仅仅使用了超级终端的部分功能来实现 PC 与芯片的串口通信。在使用“L”命令将用户 *.S19 文件下载到芯片中时，必须采用如图 15-31 所示的方法。即在键入“L”命令后，使用“传送(T)”中的“发送文本文件(S)”，弹出如图 15-32 所示窗口。

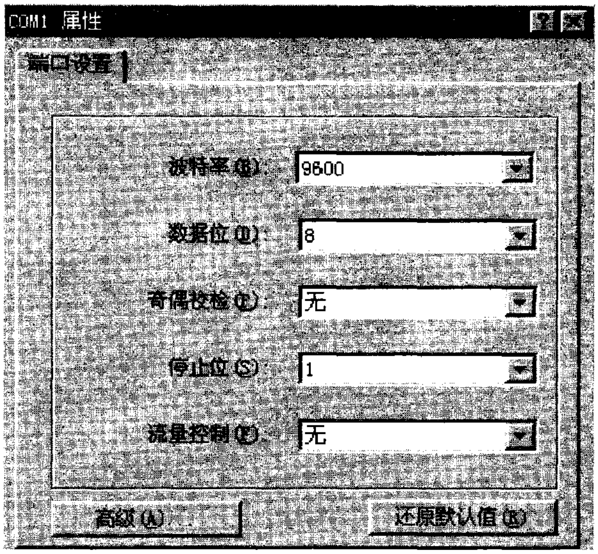


图 15-29　串口通信设置

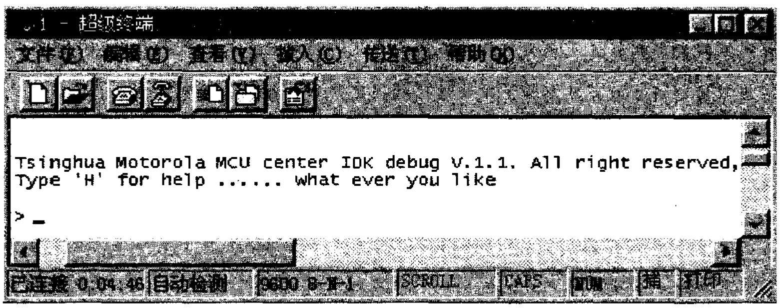


图 15-30　通信成功

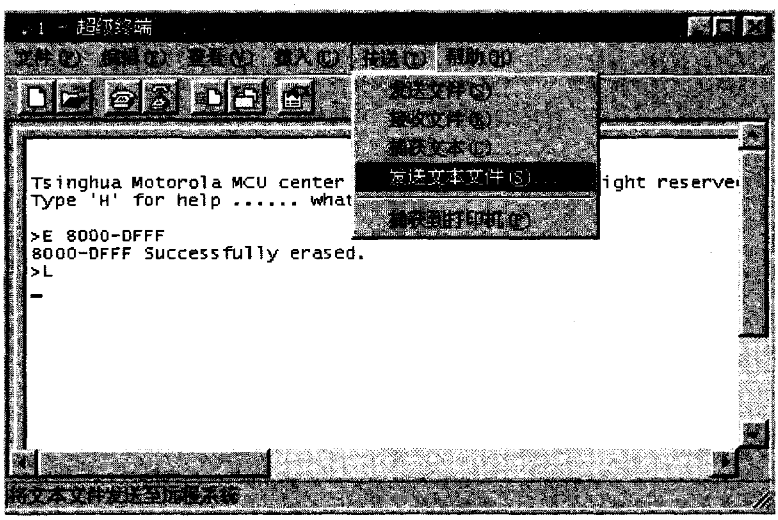


图 15-31　发送 *.s19 文件

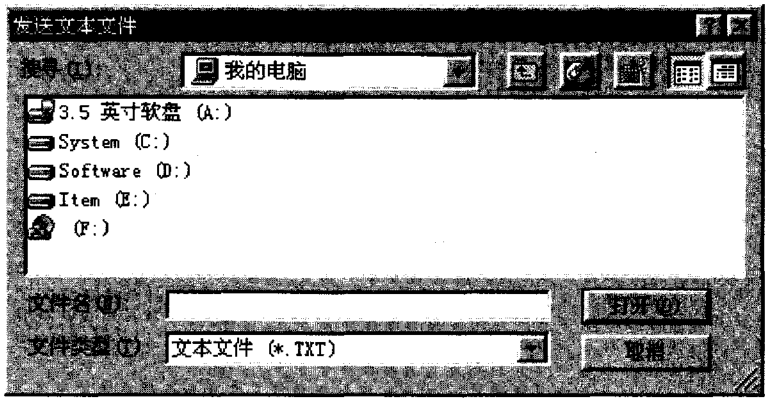
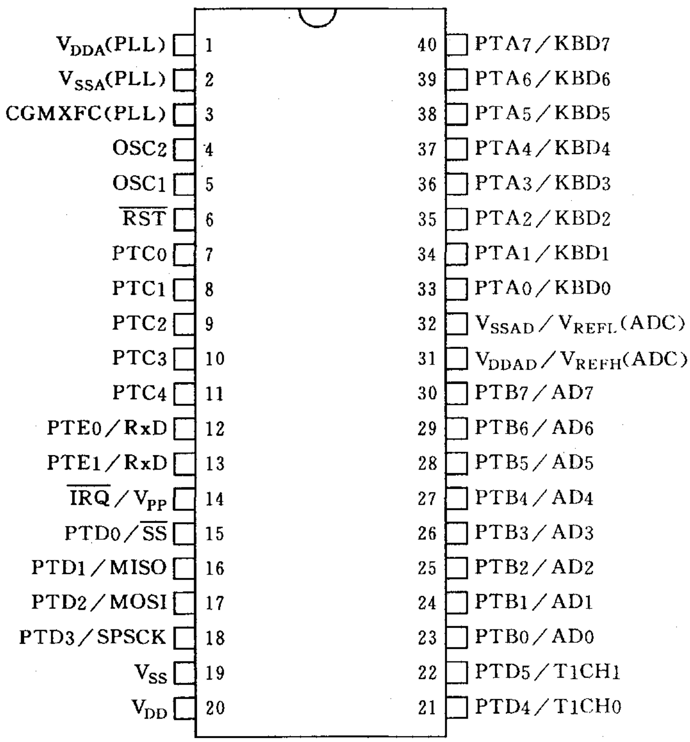


图 15-32 选择 *.S19 文件

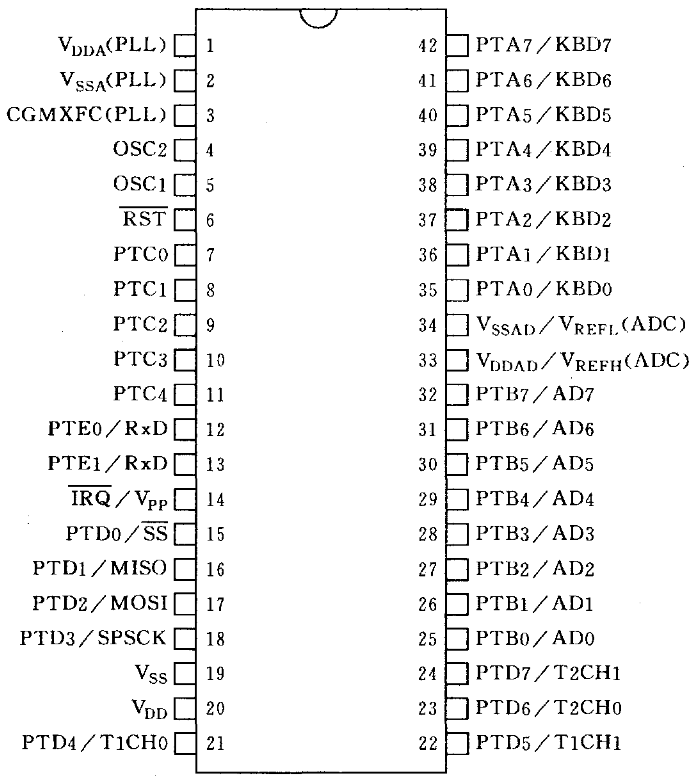
在找到要下载的 *.S19 文件后，按“打开(O)”，这样，就把文件传送到 IDK 上的 MC68HC908GP32 芯片中了。

附　　录

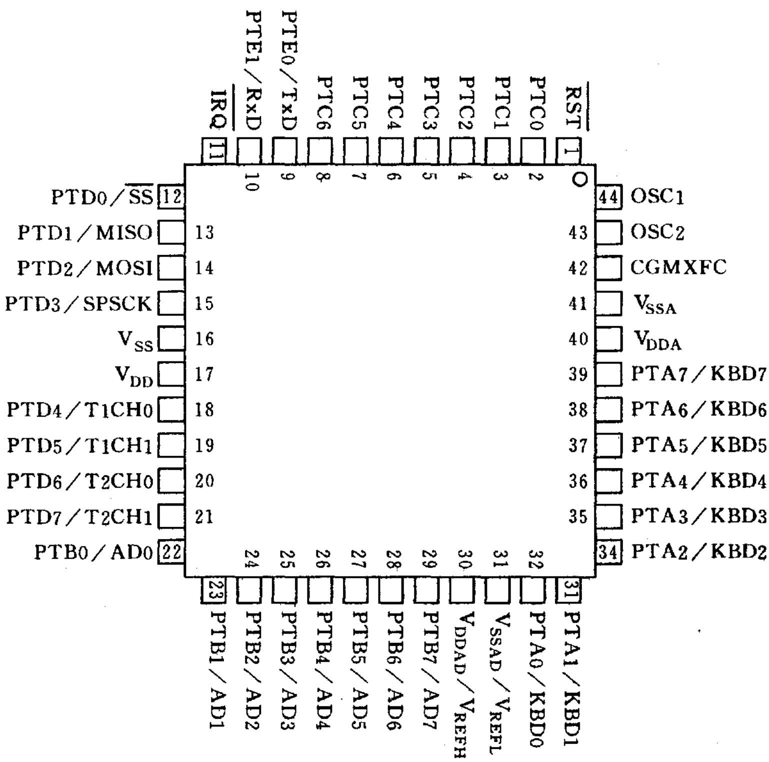
1. MC68HC908GP32 引脚图



注：此种封装的芯片没有 PTC5、PTC6、PTD6 和 PTD7 引脚。



注:此种封装的芯片没有 PTC5 和 PTC6 引脚。



RST
PTC0
PTC1
PTC2
PTC3
PTC4
PTC5
PTC6
PTE0/TxD
PTE1/RxD
IRQ
PTD0/SS
PTD1/MISO
PTD2/MOSI
PTD3/SPSCK
VSS
VDD
PTD4/T1CH0
PTD5/T1CH1
PTD6/T2CH0
PTD7/T2CH1
PTB0/AD0
PTB1/AD1
PTB2/AD2
PTB3/AD3
PTB4/AD4
PTB5/AD5
PTB6/AD6
PTB7/AD7
VDDAD/VREFH
VSSAD/VREFL
PTA0/KBD0
PTA1/KBD1
PTA2/KBD2
PTA3/KBD3
PTA4/KBD4
PTA5/KBD5
PTA6/KBD6
PTA7/KBD7
VDDA
VSSA
CGMXFC
OSC2
OSC1

2. 存储器分配图

地址	内容
$ 0000 ↓ $ 003F	I/O 寄存器 64 B
$ 0040 ↓ $ 023F	RAM 512 B
$ 0240 ↓ $ 7FFF	未存在的单元 32,192 B
$ 8000 ↓ $ FDFF	FLASH 存储器 32,256 B
$ FE00	SIM Break Status Register(SBSR)
$ FE01	SIM Reset Status Register(SRSR)
$ FE02	Reserved(SUBAR)
$ FE03	SIM Break Flag Control Register(SBFCR)
$ FE04	Interrupt Status Register 1(INT1)
$ FE05	Interrupt Status Register 2(INT2)
$ FE06	Interrupt Status Register 3(INT3)
$ FE07	保留单元
$ FE08	FLASH Control Register (FLCR)
$ FE09	Break Address Register High (BRKH)
$ FE0A	Break Address Register Low (BRKL)
$ FE0B	Break Status and Control Register (BRKSCR)
$ FE0C	LVI Status Register (LVISR)
$ FE0D ↓ $ FE0F	未存在的单元 3 B
$ FE10 ↓ $ FE1F	未存在的单元 16 B [illegible]
$ FE20 ↓ $ FF52	监控 ROM 307 B
$ FF53 ↓ $ FF7D	未存在的单元 43 B
$ FF7E	FLASH Block Protect Register (FLBPR)
$ FF7F ↓ $ FFDB	未存在的单元 93 B
$ FFDC ↓ $ FFFF	FLASH 矢量区 36 B

注：$ FFF6～$ FFFD 为 8 个保密字节。

3. 寄存器表

地址	寄存器名称		bit 7	6	5	4	3	2	1	bit 0
$ 0000	Port A Data Register (PTA)	Read: Write:	PTA7	PTA6	PTA5	PTA4	PTA3	PTA2	PTA1	PTA0
		Reset:	不受复位的影响							
$ 0001	Port B Data Register (PTB)	Read: Write:	PTB7	PTB6	PTB5	PTB4	PTB3	PTB2	PTB1	PTB0
		Reset:	不受复位的影响							
$ 0002	Port C Data Register (PTC)	Read:	0	PTC6	PTC5	PTC4	PTC3	PTC2	PTC1	PTC0
		Write:								
		Reset:	不受复位的影响							
$ 0003	Port D Data Register (PTD)	Read: Write:	PTD7	PTD6	PTD5	PTD4	PTD3	PTD2	PTD1	PTD0
		Reset:	不受复位的影响							
$ 0004	Data Direction Register A (DDRA)	Read: Write:	DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0
		Reset:	0	0	0	0	0	0	0	0
$ 0005	Data Direction Register B (DDRB)	Read: Write:	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
		Reset:	0	0	0	0	0	0	0	0
$ 0006	Data Direction Register C (DDRC)	Read:	0	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$ 0007	Data Direction Register D (DDRD)	Read: Write:	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
		Reset:	0	0	0	0	0	0	0	0
$ 0008	Port E Data Register (PTE)	Read:	0	0	0	0	0	0	PTE1	PTE0
		Write:								
		Reset:	不受复位的影响							
$ 0009	Unimplemented	Read:								
		Write:								
		Reset:	0	0	0	0	0	0	0	0

▒=不存在　　R=保留　　U=不影响

地址	寄存器名称		bit 7	6	5	4	3	2	1	bit 0
$000A	Unimplemented	Read:								
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$000B	Unimplemented	Read:								
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$000C	Data Direction Register E (DDRE)	Read:	0	0	0	0	0	0	DDRE1	DDRE0
		Write:							DDRE1	DDRE0
		Reset:	0	0	0	0	0	0	0	0
$000D	Port A Input Pullup Enable Register(PTAPUE)	Read:	PTAPUE7	PTAPUE6	PTAPUE5	PTAPUE4	PTAPUE3	PTAPUE2	PTAPUE1	PTAPUE0
		Write:	PTAPUE7	PTAPUE6	PTAPUE5	PTAPUE4	PTAPUE3	PTAPUE2	PTAPUE1	PTAPUE0
		Reset:	0	0	0	0	0	0	0	0
$000E	Port C Input Pullup Enable Register(PTCPUE)	Read:	0	PTCPUE6	PTCPUE5	PTCPUE4	PTCPUE3	PTCPUE2	PTCPUE1	PTCPUE0
		Write:		PTCPUE6	PTCPUE5	PTCPUE4	PTCPUE3	PTCPUE2	PTCPUE1	PTCPUE0
		Reset:	0	0	0	0	0	0	0	0
$000F	Port D Input Pullup Enable Register(PTDPUE)	Read:	PTDPUE7	PTDPUE6	PTDPUE5	PTDPUE4	PTDPUE3	PTDPUE2	PTDPUE1	PTDPUE0
		Write:	PTDPUE7	PTDPUE6	PTDPUE5	PTDPUE4	PTDPUE3	PTDPUE2	PTDPUE1	PTDPUE0
		Reset:	0	0	0	0	0	0	0	0
$0010	SPI Control Register (SPCR)	Read:	SPRIE	DMAS	SPMSTR	CPOL	CPHA	SPWOM	SPE	SPTIE
		Write:	SPRIE		SPMSTR	CPOL	CPHA	SPWOM	SPE	SPTIE
		Reset:	0	0	1	0	1	0	0	0
$0011	SPI Status and Control Register(SPSCR)	Read:	SPRF	ERRIE	OVRF	MODF	SPTE	MODFEN	SPR1	SPR0
		Write:		ERRIE				MODFEN	SPR1	SPR0
		Reset:	0	0	0	0	1	0	0	0
$0012	SPI Data Register (SPDR)	Read:	R7	R6	R5	R4	R3	R2	R1	R0
		Write:	T7	T6	T5	T4	T3	T2	T1	T0
		Reset:	不受复位影响							
$0013	SCI Control Register 1 (SCC1)	Read:	LOOPS	ENSCI	TXINV	M	WAKE	ILTY	PEN	PTY
		Write:	LOOPS	ENSCI	TXINV	M	WAKE	ILTY	PEN	PTY
		Reset:	0	0	0	0	0	0	0	0

▒▒=不存在　R=保留　U=不影响

地址	寄存器名称		bit 7	6	5	4	3	2	1	bit 0
$ 0014	SCI Control Register 2 (SCC2)	Read: / Write:	SCTIE	TCIE	SCRIE	ILIE	TE	RE	R… [illegible]	[illegible]
		Reset:	0	0	0	0	0	0	[illegible]	[illegible]
$ 0015	SCI Control Register 3 (SCC3)	Read:	R8	T8	DMARE	DMATE	ORIE	NEIE	F… [illegible]	[illegible]
		Write:								
		Reset:	U	U	0	0	0	[illegible]	[illegible]	[illegible]
$ 0016	SCI Status Register 1 (SCS1)	Read:	SCTE	TC	SCRF	IDLE	O… [illegible]	[illegible]	[illegible]	[illegible]
		Write:								
		Reset:	1	1	0	0	0	[illegible]	[illegible]	[illegible]
$ 0017	SCI Status Register 2 (SCS2)	Read:							[illegible]	[illegible]
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$ 0018	SCI Data Register (SCDR)	Read:	R7	R6	R5	R4	R3	R2	R1	R0
		Write:	T7	T6	T5	T4	T3	T2	T1	T0
		Reset:	不受复位影响							
$ 0019	SCI Baud Rate Register (SCBR)	Read: / Write:			SCP1	SCP0	R	SCR2	SCR1	SCR0
		Reset:	0	0	0	0	0	0	0	0
$ 001A	Keyboard Status and Control Register (INTKBSCR)	Read:	0	0	0	0	KEYF	0	IMASKK	MODEK
		Write:						ACKK		
		Reset:	0	0	0	0	0	0	0	0
$ 001B	Keyboard Interrupt Enable Register(INTKBIER)	Read: / Write:	KBIE7	KBIE6	KBIE5	KBIE4	KBIE3	KBIE2	KBIE1	KBIE0
		Reset:	0	0	0	0	0	0	0	0
$ 001C	Time Base Module Control Register(TBCR)	Read:	TBIF	TBR2	TBR1	TBR0	0	TBIE	TBON	R
		Write:					TACK			
		Reset:	0	0	0	0	0	0	0	0
$ 001D	IRQ Status and Control Register(INTSCR)	Read:	0	0	0	0	IRQF1	0	IMASK1	MODE1
		Write:						ACK1		
		Reset:	0	0	0	0	0	0	0	0

■=不存在　R=保留　U=不影响

寄存器名称			bit 7	6	5	4	3	2	1	bit 0
	…figuration Register 2 (CONFIG2)†	Read:	0	0	0	0	0	0	OSCST-OPENB	SCIBD-SRC
		Write:								
		Reset:	0	0	0	0	0	0	0	0
	…figuration Register 1 …CONFIG1)†	Read:	COPRS	LVIS-TOP	LVIR-STD	LVIP-WRD	LVI5-OR3†	SSREC	STOP	COPD
		Write:								
		Reset:	0	0	0	0	0	0	0	0
	…ontrol …TSC)	Read:	TOF	TOIE	TSTOP	0	0	PS2	PS1	PS0
		Write:	0			TRST				
		Reset:	0	0	1	0	0	0	0	0
	…Counter Register High (T1CNTH)	Read:	bit 15	14	13	12	11	10	9	bit 8
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$ 0022	Timer 1 Counter Register Low (T1CNTL)	Read:	bit 7	6	5	4	3	2	1	bit 0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$ 0023	Timer 1 Counter Modulo Register High (T1MODH)	Read:	bit 15	14	13	12	11	10	9	bit 8
		Write:								
		Reset:	1	1	1	1	1	1	1	1
$ 0024	Timer 1 Counter Modulo Register Low (T1MODL)	Read:	bit 7	6	5	4	3	2	1	bit 0
		Write:								
		Reset:	1	1	1	1	1	1	1	1
$ 0025	Timer 1 Channel 0 Status and Control Register (T1SC0)	Read:	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	TOV0	CH0MAX
		Write:	0							
		Reset:	0	0	0	0	0	0	0	0
$ 0026	Timer 1 Channel 0 Register High (T1CH0H)	Read:	bit 15	14	13	12	11	10	9	bit 8
		Write:								
		Reset:	复位后其值不确定							
$ 0027	Timer 1 Channel 0 Register Low (T1CH0L)	Read:	bit 7	6	5	4	3	2	1	bit 0
		Write:								
		Reset:	复位后其值不确定							

注：符号"†"表示：每次复位后只能写一次的寄存器（除 CONFIG1 的 LVI5OR3 位外），而 LVI5OR3 位只能通过上电复位 POR(power-on reset)才能清 0。

▒ = 不存在　　R = 保留　　U = 不影响

地址	寄存器名称		bit 7	6	5	4	3	2	1	bit 0
$ 0028	Timer 1 Channel 1 Status and Control Register (T1SC1)	Read:	CH1F	CH1IE	0	MS1A	ELS1B	ELS1A	TOV1	CH1MAX
		Write:	0							
		Reset:	0	0	0	0	0	0	0	0
$ 0029	Timer 1 Channel 1 Register High (T1CH1H)	Read:	bit 15	14	13	12	11	10	9	bit 8
		Write:								
		Reset:	复位后其值不确定							
$ 002A	Timer 1 Channel 1 Register Low (T1CH1L)	Read:	bit 7	6	5	4	3	2	1	bit 0
		Write:								
		Reset:	复位后其值不确定							
$ 002B	Timer 2 Status and Control Register (T2SC)	Read:	TOF	TOIE	TSTOP	0	0	PS2	PS1	PS0
		Write:	0			TRST				
		Reset:	0	0	1	0	0	0	0	0
$ 002C	Timer 2 Counter Register High (T2CNTH)	Read:	bit 15	14	13	12	11	10	9	bit 8
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$ 002D	Timer 2 Counter Register Low (T2CNTL)	Read:	bit 7	6	5	4	3	2	1	bit 0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$ 002E	Timer 2 Counter Modulo Register High (T2MODH)	Read:	bit 15	14	13	12	11	10	9	bit 8
		Write:								
		Reset:	1	1	1	1	1	1	1	1
$ 002F	Timer 2 Counter Modulo Register Low (T2MODL)	Read:	bit 7	6	5	4	3	2	1	bit 0
		Write:								
		Reset:	1	1	1	1	1	1	1	1
$ 0030	Timer 2 Channel 0 Status and Control Register (T2SC0)	Read:	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	TOV0	CH0MAX
		Write:	0							
		Reset:	0	0	0	0	0	0	0	0
$ 0031	Timer 2 Channel 0 Register High (T2CH0H)	Read:	bit 15	14	13	12	11	10	9	bit 8
		Write:								
		Reset:	复位后其值不确定							

(阴影)=不存在　　R=保留　　U=不影响

地址	寄存器名称		bit 7	6	5	4	3	2	1	bit 0
$0032	Timer 2 Channel 0 Register Low (T2CH0L)	Read/Write:	bit 7	6	5	4	3	2	1	bit 0
		Reset:	复位后其值不确定							
$0033	Timer 2 Channel 1 Status and Control Register (T2SC1)	Read:	CH1F	CH1IE	0	MS1A	ELS1B	ELS1A	TOV1	CH1MAX
		Write:	0	CH1IE		MS1A	ELS1B	ELS1A	TOV1	CH1MAX
		Reset:	0	0	0	0	0	0	0	0
$0034	Timer 2 Channel 1 Register High (T2CH1H)	Read/Write:	bit 15	14	13	12	11	10	9	bit 8
		Reset:	复位后其值不确定							
$0035	Timer 2 Channel 1 Register Low (T2CH1L)	Read/Write:	bit 7	6	5	4	3	2	1	bit 0
		Reset:	复位后其值不确定							
$0036	PLL Control Register (PCTL)	Read:	PLLIE	PLLF	PLLON	BCS	PRE1	PRE0	VPR1	VPR0
		Write:	PLLIE		PLLON	BCS	PRE1	PRE0	VPR1	VPR0
		Reset:	0	0	1	0	0	0	0	0
$0037	PLL Bandwidth Control Register (PBWC)	Read:	AUTO	LOCK	$\overline{ACQ}$	0	0	0	0	R
		Write:	AUTO		$\overline{ACQ}$					R
		Reset:	0	0	0	0	0	0	0	0
$0038	PLL Multiplier Select High Register (PMSH)	Read:	0	0	0	0	MUL11	MUL10	MUL9	MUL8
		Write:					MUL11	MUL10	MUL9	MUL8
		Reset:	0	0	0	0	0	0	0	0
$0039	PLL Multiplier Select Low Register (PMSL)	Read/Write:	MUL7	MUL6	MUL5	MUL4	MUL3	MUL2	MUL1	MUL0
		Reset:	0	1	0	0	0	0	0	0
$003A	PLL VCO Select Range Register (PMRS)	Read/Write:	VRS7	VRS6	VRS5	VRS4	VRS3	VRS2	VRS1	VRS0
		Reset:	0	1	0	0	0	0	0	0
$003B	PLL Reference Divider Select Register (PMDS)	Read:	0	0	0	0	RDS3	RDS2	RDS1	RDS0
		Write:					RDS3	RDS2	RDS1	RDS0
		Reset:	0	0	0	0	0	0	0	1

(阴影)=不存在　　R=保留　　U=不影响

地址	寄存器名称		bit 7	6	5	4	3	2	1	bit 0
$ 003C	Analog-to-Digital Status and Control Register (ADSCR)	Read:	COCO	AIEN	ADCO	ADCH4	ADCH3	ADCH2	ADCH1	ADCH0
		Write:	R							
		Reset:	0	0	0	1	1	1	1	1
$ 003D	Analog-to-Digital Data Register (ADR)	Read:	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
		Write:	R	R	R	R	R	R	R	R
		Reset:	复位后其值不确定							
$ 003E	Analog-to-Digital Input Clock Register (ADCLK)	Read:	ADIV2	ADIV1	ADIV0	ADICLK	0	0	0	0
		Write:					R	R	R	R
		Reset:	0	0	0	0	0	0	0	0
$ 003F	Unimplemented	Read:								
		Write:								
		Reset:								
$ FE00	SIM Break Status Register (SBSR)	Read:	R	R	R	R	R	R	SBSW	R
		Write:							NOTE	
		Reset:	0	0	0	0	0	0	0	0

Note: Writing a logic 0 clears SBSW.

地址	寄存器名称		bit 7	6	5	4	3	2	1	bit 0
$ FE01	SIM Reset Status Register (SBSR)	Read:	POR	PIN	COP	ILOP	ILAD	MODRST	LVI	0
		Write:								
		Reset:	1	0	0	0	0	0	0	0
$ FE02	SIM Upper Byte Address Register (SUBAR)	Read:	R	R	R	R	R	R	R	R
		Write:								
		Reset:								
$ FE03	SIM Break Flag Control Register (SBFCR)	Read:	BCFE	R	R	R	R	R	R	R
		Write:								
		Reset:	0							
$ FE04	Interrupt Status Register 1 (INT 1)	Read:	IF6	IF5	IF4	IF3	IF2	IF1	0	0
		Write:	R	R	R	R	R	R	R	R
		Reset:	0	0	0	0	0	0	0	0
$ FE05	Interrupt Status Register 2 (INT 2)	Read:	IF14	IF13	IF12	IF11	IF10	IF9	IF8	IF7
		Write:	R	R	R	R	R	R	R	R
		Reset:	0	0	0	0	0	0	0	0

▓▓ =不存在　　R=保留　　U=不影响

地址	寄存器名称		bit 7	6	5	4	3	2	1	bit 0
$FE06	Interrupt Status Register 3 (INT 3)	Read:	0	0	0	0	0	0	IF16	IF15
		Write:	R	R	R	R	R	R	R	R
		Reset:	0	0	0	0	0	0	0	0
$FE07	Reserved	Read:	R	R	R	R	R	R	R	R
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$FE08	FLASH Control Register (FLCR)	Read:	0	0	0	0	HVEN	MASS	ERASE	PGM
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$FE09	Break Address Register High (BRKH)	Read:	bit 15	14	13	12	11	10	9	bit 8
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$FE0A	Break Address Register Low (BRKL)	Read:	bit 7	6	5	4	3	2	1	bit 0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$FE0B	Break Status and Control Register (BRKSCR)	Read:	BRKE	BRKA	0	0	0	0	0	0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$FE0C	LVI Status Register (LVISR)	Read:	LVIOUT	0	0	0	0	0	0	0
		Write:								
		Reset:	0	0	0	0	0	0	0	0
$FF7E	FLASH Block Protect Register (FLBPR)†	Read:	BPR7	BPR6	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0
		Write:								
		Reset:	U	U	U	U	U	U	U	U
$FFFF	COP Control Register (COPCTL)	Read:	读时为复位向量的低字节							
		Write:	写任意值则清除 COP 计数器							
		Reset:	不受复位影响							

注:符号"†"表示该寄存器为非易失 Flash 寄存器。

▒▒ =不存在　　R=保留　　U=不影响

4. 中断矢量表

中断优先级	中断寄存器中的标志位	中断矢量地址	中断源
最低 ↑	IF16	$FFDC	时基(高)
		$FFDD	时基(低)
	IF15	$FFDE	ADC 转换结束(高)
		$FFDF	ADC 转换结束(低)
	IF14	$FFE0	键盘(高)
		$FFE1	键盘(低)
	IF13	$FFE2	SCI 发送(高)
		$FFE3	SCI 发送(低)
	IF12	$FFE4	SCI 接收(高)
		$FFE5	SCI 接收(低)
	IF11	$FFE6	SCI 错误(高)
		$FFE7	SCI 错误(低)
	IF10	$FFE8	SPI 发送(高)
		$FFE9	SPI 发送(低)
	IF9	$FFEA	SPI 接收(高)
		$FFEB	SPI 接收(低)
	IF8	$FFEC	定时器 2 溢出(高)
		$FFED	定时器 2 溢出(低)
	IF7	$FFEE	定时器 2 通道 1(高)
		$FFEF	定时器 2 通道 1(低)
	IF6	$FFF0	定时器 2 通道 0(高)
		$FFF1	定时器 2 通道 0(低)
	IF5	$FFF2	定时器 1 溢出(高)
		$FFF3	定时器 1 溢出(低)
	IF4	$FFF4	定时器 1 通道 1(高)
		$FFF5	定时器 1 通道 1(低)
	IF3	$FFF6	定时器 1 通道 0(高)
		$FFF7	定时器 1 通道 0(低)
	IF2	$FFF8	锁相环(高)
		$FFF9	锁相环(低)
	IF1	$FFFA	$\overline{IRQ}$引脚(高)
		$FFFB	$\overline{IRQ}$引脚(低)
	—	$FFFC	软中断指令(高)
		$FFFD	软中断指令(低)
	—	$FFFE	复位(高)
↓ 最高		$FFFF	复位(低)

5. MC68HC08 指令集操作表

	位操作		转移	读一修改一写						控制		寄存器/存储器							
	DIR	DIR	REL	DIR	INH	INH	IX1	SP1	IX	INH	INH	IMM	DIR	EXT	IX2	SP2	IX1	SP1	IX
高 / 低	0	1	2	3	4	5	6	9E6	7	8	9	A	B	C	D	9ED	E	9EE	F
0	BRSET0 5 3 DIR	BSET0 4 2 DIR	BRA 3 2 REL	NEG 4 2 DIR	NEGA 1 1 INH	NEGX 1 1 INH	NEG 4 2 IX1	NEG 5 3 SP1	NEG 3 1 IX	RTI 7 1 INH	BGE 3 2 REL	SUB 2 2 IMM	SUB 3 2 DIR	SUB 4 3 EXT	SUB 4 3 IX2	SUB 5 4 SP2	SUB 3 2 IX1	SUB 4 3 SP1	SUB 2 1 IX
1	BRCLR0 5 3 DIR	BCLR0 4 2 DIR	BRN 3 2 REL	CBEQ 5 3 DIR	CBEQA 4 3 IMM	CBEQX 4 3 IMM	CBEQ 5 3 IX1+	CBEQ 6 4 SP1	CBEQ 4 2 IX+	RTS 4 1 INH	BLT 3 2 REL	CMP 2 2 IMM	CMP 3 2 DIR	CMP 4 3 EXT	CMP 4 3 IX2	CMP 5 4 SP2	CMP 3 2 IX1	CMP 4 3 SP1	CMP 2 1 IX
2	BRSET1 5 3 DIR	BSET1 4 2 DIR	BHI 3 2 REL		MUL 5 1 INH	DIV 7 1 INH	NSA 3 1 INH		DAA 2 1 INH		BGT 3 2 REL	SBC 2 2 IMM	SBC 3 2 DIR	SBC 4 3 EXT	SBC 4 3 IX2	SBC 5 4 SP2	SBC 3 2 IX1	SBC 4 3 SP1	SBC 2 1 IX
3	BRCLR1 5 3 DIR	BCLR1 4 2 DIR	BLS 3 2 REL	COM 4 2 DIR	COMA 1 1 INH	COMX 1 1 IX1	COM 4 2 IX1	COM 5 3 SP1	COM 3 1 IX	SWI 9 1 INH	BLE 3 2 REL	CPX 2 2 IMM	CPX 3 2 DIR	CPX 4 3 EXT	CPX 4 3 IX2	CPX 5 4 SP2	CPX 3 2 IX1	CPX 4 3 SP1	CPX 2 1 IX
4	BRSET2 5 3 DIR	BSET2 4 2 DIR	BCC 3 2 REL	LSR 4 2 DIR	LSRA 1 1 INH	LSRX 1 1 IXH	LSR 4 2 IX1	LSR 5 3 SP1	LSR 3 1 IX	TAP 2 1 INH	TXS 2 1 INH	AND 2 2 IMM	AND 3 2 DIR	AND 4 3 EXT	AND 4 3 IX2	AND 5 4 SP2	AND 3 2 IX1	AND 4 3 SP1	AND 2 1 IX
5	BRCLR2 5 3 DIR	BCLR2 4 2 DIR	BCS 3 2 REL	STHX 4 2 DIR	LDHX 3 3 IMM	LDHX 4 2 DIR	CPHX 3 3 IMM		CPHX 4 2 DIR	TPA 1 1 INH	TSX 2 1 INH	BIT 2 2 IMM	BIT 3 2 DIR	BIT 4 3 EXT	BIT 4 3 IX2	BIT 5 4 SP2	BIT 3 2 IX1	BIT 4 3 SP1	BIT 2 1 IX
6	BRSET3 5 3 DIR	BSET3 4 2 DIR	BNE 3 2 REL	ROR 4 2 DIR	RORA 1 1 INH	RORX 1 1 INH	ROR 4 2 IX1	ROR 5 3 SP1	ROR 3 1 IX	PULA 2 1 INH		LDA 2 2 IMM	LDA 3 2 DIR	LDA 4 3 EXT	LDA 4 3 IX2	LDA 5 4 SP2	LDA 3 2 IX1	LDA 4 3 SP1	LDA 2 1 IX
7	BRCLR3 5 3 DIR	BCLR3 4 2 DIR	BEQ 3 2 REL	ASR 4 2 DIR	ASRA 1 1 INH	ASRX 1 1 INH	ASR 4 2 IX1	ASR 5 3 SP1	ASR 3 1 IX	PSHA 2 1 INH	TAX 1 1 INH	AIS 2 2 IMM	STA 3 2 DIR	STA 4 3 EXT	STA 4 3 IX2	STA 5 4 SP2	STA 3 2 IX1	STA 4 3 SP1	STA 2 1 IX
8	BRSET4 5 3 DIR	BSET4 4 2 DIR	BHCC 3 2 REL	LSL 4 2 DIR	LSLA 1 1 INH	LSLX 1 1 INH	LSL 4 2 IX1	LSL 5 3 SP1	LSL 3 1 IX	PULX 2 1 INH	CLC 1 1 INH	EOR 2 2 IMM	EOR 3 2 DIR	EOR 4 3 EXT	EOR 4 3 IX2	EOR 5 4 SP2	EOR 3 2 IX1	EOR 4 3 SP1	EOR 2 1 IX

续表

	位操作		转移	读—修改—写						控制		寄存器/存储器							
	DIR	DIR	REL	DIR	INH	INH	IX1	SP1	IX	INH	INH	IMM	DIR	EXT	IX2	SP2	IX1	SP1	IX
低＼高	0	1	2	3	4	5	6	9E6	7	8	9	A	B	C	D	9ED	E	9EE	F
9	BRCLR4 5 3 DIR	BCLR4 4 2 DIR	BHCS 3 2 REL	ROL 4 2 DIR	ROLA 1 1 INH	ROLX 1 1 INH	ROL 4 2 IX1	ROL 5 3 SP1	ROL 3 1 IX	PSHX 2 1 INH	SEC 1 1 INH	ADC 2 2 IMM	ADC 3 2 DIR	ADC 4 3 EXT	ADC 4 3 IX2	ADC 5 4 SP2	ADC 3 2 IX1	ADC 4 3 SP1	ADC 2 1 IX
A	BRSET5 5 3 DIR	BSET5 4 2 DIR	BPL 3 2 REL	DEC 4 2 DIR	DECA 1 1 INH	DECX 1 1 INH	DEC 4 2 IX1	DEC 5 3 SP1	DEC 3 1 IX	PULH 2 1 INH	CLI 2 1 INH	ORA 2 2 IMM	ORA 3 2 DIR	ORA 4 3 EXT	ORA 4 3 IX2	ORA 5 4 SP2	ORA 3 2 IX1	ORA 4 3 SP1	ORA 2 1 IX
B	BRCLR5 5 3 DIR	BCLR5 4 2 DIR	BMI 3 2 REL	DBNZ 5 3 DIR	DBNZA 3 2 INH	DBNZX 3 2 INH	DBNZ 5 3 IX1	DBNZ 6 4 SP1	DBNZ 4 2 IX	PSHH 2 1 INH	SEI 2 1 INH	ADD 2 2 IMM	ADD 3 2 DIR	ADD 4 3 EXT	ADD 4 3 IX2	ADD 5 4 SP2	ADD 3 2 IX1	ADD 4 3 SP1	ADD 2 1 IX
C	BRSET6 5 3 DIR	BSET6 4 2 DIR	BMC 3 2 REL	INC 4 2 DIR	INCA 1 1 INH	INCX 1 1 INH	INC 4 2 IX1	INC 5 3 SP1	INC 3 1 IX	CLRH 1 1 INH	RSP 1 1 INH		JMP 2 2 DIR	JMP 3 3 EXT	JMP 4 3 IX2		JMP 3 2 IX1		JMP 2 1 IX
D	BRCLR6 5 3 DIR	BCLR6 4 2 DIR	BMS 3 2 REL	TST 3 2 DIR	TSTA 1 1 INH	TSTX 1 1 INH	TST 3 2 IX1	TST 4 3 SP1	TST 2 1 IX		NOP 1 1 INH	BSR 4 2 REL	JSR 4 2 DIR	JSR 5 3 EXT	JSR 6 3 IX2		JSR 5 2 IX1		JSR 4 1 IX
E	BRSET7 5 3 DIR	BSET7 4 2 DIR	BIL 3 2 REL		MOV 5 3 DD	MOV 4 2 DIX+	MOV 4 3 IMD		MOV 4 2 IX+D	STOP 1 1 INH	*	LDX 2 2 IMM	LDX 3 2 DIR	LDX 4 3 EXT	LDX 4 3 IX2	LDX 5 4 SP2	LDX 3 2 IX1	LDX 4 3 SP1	LDX 2 1 IX
F	BRCLR7 5 3 DIR	BCLR7 4 2 DIR	BIH 3 2 REL	CLR 3 2 DIR	CLRA 1 1 INH	CLRX 1 1 INH	CLR 3 2 IX1	CLR 4 3 SP1	CLR 2 1 IX	WAIT 1 1 INH	TXA 1 1 INH	AIX 2 2 IMM	STX 3 2 DIR	STX 4 3 EXT	STX 4 3 IX2	STX 5 4 SP2	STX 3 2 IX1	STX 4 3 SP1	STX 2 1 IX

INH	隐含	REL	相对	SP1	8 位偏移量堆栈指针
IMM	立即	IX	无偏移量变址	SP2	16 位偏移量堆栈指针
DIR	直接	IX1	8 位偏移量变址	IX+	无偏移量变址预加 1
EXT	扩展	IX2	16 位偏移量变址	IX1+	1 字节偏移量变址预加 1
DD	直接到直接	IMD	立即到直接		
IX+D	变址到直接	DIX+	直接到变址		

16 进制操作码的高位字节 F

16 进制操作码的低位字节 0 — SUB 2
1 IX

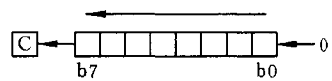
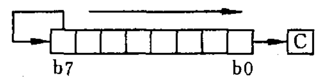
HC08 周期数
操作码助记符
字节数/寻址方式

6. MC68HC08 指令系统详细说明

中文说明

指令助记符	操　　作	功能描述	条件码 V	H	I	N	Z	C	寻址方式	操作码	操作数	周期数
ADC #opr	带进位加	A←(A)+(M)+(C)	↕	↕	—	↕	↕	↕	IMM	A9	ii	2
ADC opr									DIR	B9	dd	3
ADC opr									EXT	C9	hh ll	4
ADC opr,X									IX2	D9	ee ff	4
ADC opr,X									IX1	E9	ff	3
ADC ,X									IX	F9		2
ADC opr,SP									SP1	9EE9	ff	4
ADC opr,SP									SP2	9ED9	ee ff	5
ADD #opr	不带进位加	A←(A)+(M)	↕	↕	—	↕	↕	↕	IMM	AB	ii	2
ADD opr									DIR	BB	dd	3
ADD opr									EXT	CB	hh ll	4
ADD opr,X									IX2	DB	ee ff	4
ADD opr,X									IX1	EB	ff	3
ADD ,X									IX	FB		2
ADD opr,SP									SP1	9EEB	ff	4
ADD opr,SP									SP2	9EDB	ee ff	5
AIS #opr	有符号立即数加到堆栈指针	SP←(SP)+(16«M)	—	—	—	—	—	—	IMM	A7	ii	2
AIX.#opr	立即数加到变址寄存器	H:X←(H:X)+(16«M)	—	—	—	—	—	—	IMM	AF	ii	2
AND #opr	逻辑"与"	A←(A)和(M)与运算的结果	0	—	—	↕	↕	—	IMM	A4	ii	2
AND opr									DIR	B4	dd	3
AND opr									EXT	C4	hh ll	4
AND opr,X									IX2	D4	ee ff	4
AND opr,X									IX1	E4	ff	3
AND ,X									IX	F4		2
AND opr,SP									SP1	9EE4	ff	4
AND opr,SP									SP2	9ED4	ee ff	5

续表

指令助记符	操　作	功能描述	条件码 V	H	I	N	Z	C	寻址方式	操作码	操作数	周期数
ASL opr ASLA ASLX ASL opr,X ASL ,X ASL opr,SP	算术左移		↕	—	—	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5
ASR opr ASRA ASRX ASR opr,X ASR opr,X ASR opr,SP	算术右移		↕	—	—	↕	↕	↕	DIR INH INH IX1 IX SP1	37 47 57 67 77 9E67	dd ff ff	4 1 1 4 3 5
BCC rel	进位标志为 0 则转移	PC←(PC)+＄0002+rel?（C)=0	—	—	—	—	—	—	REL	24	rr	3
BCLR n,opr	清存储器第 n 位	Mn←0	—	—	—	—	—	—	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	11 13 15 17 19 1B 1D 1F	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BCS rel	进位标志为 1 则转移	PC←(PC)+＄0002+rel?（C)=1	—	—	—	—	—	—	REL	25	rr	3
BEQ rel	相等则转移	PC←(PC)+＄0002+rel?（Z)=1	—	—	—	—	—	—	REL	27	rr	3

续表

指令助记符	操 作	功 能 描 述	条件码						寻址方式	操作码	操作数	周期数
			V	H	I	N	Z	C				
BGE opr	大于或等于则转移(有符号数)	PC←(PC)+＄0002+rel? (N⊕V)=0	—	—	—	—	—	—	REL	90	rr	3
BGT opr	大于则转移(有符号数)	PC←(PC)+＄0002+rel? (Z)|(N⊕V)=0	—	—	—	—	—	—	REL	92	rr	3
BHCC rel	半进位标志为0则转移	PC←(PC)+＄0002+rel? (H)=0	—	—	—	—	—	—	REL	28	rr	3
BHCS rel	半进位标志为1则转移	PC←(PC)+＄0002+rel? (H)=1	—	—	—	—	—	—	REL	29	rr	3
BHI rel	大于则转移	PC←(PC)+＄0002+rel? (C)|(Z)=0	—	—	—	—	—	—	REL	22	rr	3
BHS rel	大于或等于则转移	PC←(PC)+＄0002+rel? (C)=0	—	—	—	—	—	—	REL	24	rr	3
BIH rel	$\overline{IRQ}$引脚为高则转移	PC←(PC)+＄0002+rel? $\overline{IRQ}$=1	—	—	—	—	—	—	REL	2F	rr	3
BIL rel	$\overline{IRQ}$引脚为低则转移	PC←(PC)+＄0002+rel? $\overline{IRQ}$=0	—	—	—	—	—	—	REL	2E	rr	3
BIT #opr	位测试	(A)和(M)与运算	0	—	—	↕	↕	—	IMM	A5	ii	2
BIT opr									DIR	B5	dd	3
BIT opr									EXT	C5	hh ll	4
BIT opr,X									IX2	D5	ee ff	4
BIT opr,X									IX1	E5	ff	3
BIT,X									IX	F5		2
BIT opr,SP									SP1	9EE5	ff	4
BIT opr,SP									SP2	9ED5	ee ff	5
BLE opr	小于或等于则转移(有符号数)	PC←(PC)+＄0002+rel? (Z)|(N⊕V)=1	—	—	—	—	—	—	REL	93	rr	3
BLO rel	小于则转移	PC←(PC)+＄0002+rel? (C)=1	—	—	—	—	—	—	REL	25	rr	3
BLS rel	小于等于则转移	PC←(PC)+＄0002+rel? (C)|(Z)=1	—	—	—	—	—	—	REL	23	rr	3
BLT opr	小于则转移(有符号数)	PC←(PC)+＄0002+rel? (N⊕V)=1	—	—	—	—	—	—	REL	91	rr	3

续表

指令助记符	操作	功能描述	条件码						寻址方式	操作码	操作数	周期数
			V	H	I	N	Z	C				
BMC rel	中断屏蔽位为 0 则转移	PC←(PC)＋＄0002＋rel?（I)＝0	—	—	—	—	—	—	REL	2C	rr	3
BMI rel	负值则转移	PC←(PC)＋＄0002＋rel?（N)＝1	—	—	—	—	—	—	REL	2B	rr	3
BMS rel	中断屏蔽位为 1 则转移	PC←(PC)＋＄0002＋rel?（I)＝1	—	—	—	—	—	—	REL	2D	rr	3
BNE rel	不等于则转移	PC←(PC)＋＄0002＋rel?（Z)＝0	—	—	—	—	—	—	REL	26	rr	3
BPL rel	正值则转移	PC←(PC)＋＄0002＋rel?（N)＝0	—	—	—	—	—	—	REL	2A	rr	3
BRA rel	无条件转移	PC←(PC)＋＄0002＋rel	—	—	—	—	—	—	REL	20	rr	3
BRCLR n，opr，rel	存储器第 n 位为 0 则转移	PC←(PC)＋＄0003＋rel?（Mn)＝0	—	—	—	—	—	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 03 05 07 09 0B 0D 0F	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BRN rel	跳至下一条指令	PC←(PC)＋＄0002	—	—	—	—	—	—	REL	21	rr	3
BRSET n，opr，rel	存储器第 n 位为 1 则转移	PC←(PC)＋＄0003＋rel?（Mn)＝1	—	—	—	—	—	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 02 04 06 08 0A 0C 0E	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5

续表

指令助记符	操　作	功能描述	条件码 V	H	I	N	Z	C	寻址方式	操作码	操作数	周期数
BSET n,opr	置存储器第 n 位为 1	Mn←1	—	—	—	—	—	—	DIR (b0)	10	dd	4
									DIR (b1)	12	dd	4
									DIR (b2)	14	dd	4
									DIR (b3)	16	dd	4
									DIR (b4)	18	dd	4
									DIR (b5)	1A	dd	4
									DIR (b6)	1C	dd	4
									DIR (b7)	1E	dd	4
BSR rel	跳至子程序地址	PC←(PC)+$0002;断点低字节入栈 SP←(SP)−$0001;断点高字节入栈 SP←(SP)−$0001 PC←(PC)+rel	—	—	—	—	—	—	REL	AD	rr	4
CBEQ opr,rel	比较,相等则转移	PC←(PC)+$0003+rel? (A)−(M)=$00	—	—	—	—	—	—	DIR	31	dd rr	5
CBEQA #opr,rel		PC←(PC)+$0003+rel? (A)−(M)=$00							IMM	41	ii rr	4
CBEQX #opr,rel		PC←(PC)+$0003+rel? (X)−(M)=$00							IMM	51	ii rr	4
CBEQ opr,X+,rel		PC←(PC)+$0003+rel? (A)−(M)=$00							IX1+	61	ff rr	5
CBEQ X+,rel		PC←(PC)+$0002+rel? (A)−(M)=$00							IX+	71	rr	4
CBEQ opr,SP,rel		PC←(PC)+$0004+rel? (A)−(M)=$00							SP1	9E61	ff rr	6
CLC	清进位标志	C←0	—	—	—	—	—	0	INH	98		1
CLI	清中断屏蔽位	I←0	—	—	0	—	—	—	INH	9A		2
CLR opr	清零指令	M←$00	0	—	—	0	1	—	DIR	3F	dd	3
CLRA		A←$00							INH	4F		1
CLRX		X←$00							INH	5F		1
CLRH		H←$00							INH	8C		1
CLR opr,X		M←$00							IX1	6F	ff	3
CLR ,X		M←$00							IX	7F		2
CLR opr,SP		M←$00							SP1	9E6F	ff	4

续表

指令助记符	操　作	功能描述	条件码 V	H	I	N	Z	C	寻址方式	操作码	操作数	周期数
CMP #opr	比较累加器与存储器	(A)−(M)	↕	—	—	↕	↕	↕	IMM	A1	ii	2
CMP opr									DIR	B1	dd	3
CMP opr									EXT	C1	hh ll	4
CMP opr,X									IX2	D1	ee ff	4
CMP opr,X									IX1	E1	ff	3
CMP ,X									IX	F1		2
CMP opr,SP									SP1	9EE1	ff	4
CMP opr,SP									SP2	9ED1	ee ff	5
COM opr	求反指令	M←($\overline{M}$)=$FF−(M)	0	—	—	↕	↕	1	DIR	33	dd	4
COMA		A←($\overline{A}$)=$FF−(M)							INH	43		1
COMX		X←($\overline{X}$)=$FF−(M)							INH	53		1
COM opr,X		M←($\overline{M}$)=$FF−(M)							IX1	63	ff	4
COM ,X		M←($\overline{M}$)=$FF−(M)							IX	73		3
COM opr,SP		M←($\overline{M}$)=$FF−(M)							SP1	9E63	ff	5
CPHX #opr	比较变址寄存器与存储器	(H:X)−(M:M+$0001)	↕	—	—	↕	↕	↕	IMM	65	ii ii+1	3
CPHX opr									DIR	75	dd	4
CPX #opr	比较X寄存器与存储器	(X)−(M)	↕	—	—	↕	↕	↕	IMM	A3	ii	2
CPX opr									DIR	B3	dd	3
CPX opr									EXT	C3	hh ll	4
CPX ,X									IX2	D3	ee ff	4
CPX opr,X									IX1	E3	ff	3
CPX opr,X									IX	F3		2
CPX opr,SP									SP1	9EE3	ff	4
CPX opr,SP									SP2	9ED3	ee ff	5
DAA	十进位转换	$(A)_{10}$	U	—	—	↕	↕	↕	INH	72		2

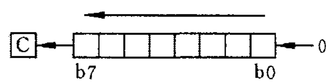
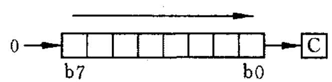
续表

指令助记符	操作	功能描述	条件码 V	H	I	N	Z	C	寻址方式	操作码	操作数	周期数
DBNZ opr,rel DBNZA rel DBNZX rel DBNZ opr,X,rel DBNZ X,rel DBNZ opr,SP,rel	减去1后结果不为0则转移	A←(A)－$0001 or M←(M)－$01 or X←(X)－$0001 PC←(PC)＋$0003＋rel 如果结果不等于零（对于 DBNZ direct,IX1 的情况） PC←(PC)＋$0002＋rel 如果结果不等于零（对于 DBNZA,DBNZX,或 IX 的情况） PC←(PC)＋$0004＋rel 如果结果不等于零（对于 DBNZ SP1 的情况）	—	—	—	—	—	—	DIR INH INH IX1 IX SP1	3B 4B 5B 6B 7B 9E6B	dd rr rr rr ff rr rr ff rr	5 3 3 5 4 6
DEC opr DECA DECX DEC opr,X DEC ,X DEC opr,SP	减1指令	M←(M)－$01 A←(A)－$01 X←(X)－$01 M←(M)－$01 M←(M)－$01 M←(M)－$01	↕	—	—	↕	↕	—	DIR INH INH IX1 IX SP1	3A 4A 5A 6A 7A 9E6A	dd ff ff	4 1 1 4 3 5
DIV	除法指令	A←(H:A)/(X) H←余数	—	—	—	—	↕	↕	INH	52		7
EOR #opr EOR opr EOR opr EOR opr,X EOR opr,X EOR ,X EOR opr,SP EOR opr,SP	存储器与累加器进行“异或”	A←(A⊕M)	0	—	—	↕	↕	—	IMM DIR EXT IX2 IX1 IX SP1 SP2	A8 B8 C8 D8 E8 F8 9EE8 9ED8	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

续表

指令助记符	操　　作	功 能 描 述	条件码 V	H	I	N	Z	C	寻址方式	操作码	操作数	周期数
INC opr INCA INCX INC opr,X INC ,X INC opr,SP	加 1 指令	M←(M)+＄01 A←(A)+＄01 X←(X)+＄01 M←(M)+＄01 M←(M)+＄01 M←(M)+＄01	↕	—	—	↕	↕	—	DIR INH INH IX1 IX SP1	3C 4C 5C 6C 7C 9E6C	dd ff ff	4 1 1 4 3 5
JMP opr JMP opr JMP opr,X JMP opr,X JMP ,X	跳转指令	PC←跳转地址	—	—	—	—	—	—	DIR EXT IX2 IX1 IX	BC CC DC EC FC	dd hh ll ee ff ff 	2 3 4 3 2
JSR opr JSR opr JSR opr,X JSR opr,X JSR ,X	跳至子程序	PC←(PC)+n(n=1,2,or 3) 断点低 8 位入栈;SP←(SP)−＄0001 断点高 8 位入栈;SP←(SP)−＄0001 PC←子程序入口地址	—	—	—	—	—	—	DIR EXT IX2 IX1 IX	BD CD DD ED FD	dd hh ll ee ff ff 	4 5 6 5 4
LDA #opr LDA opr LDA opr LDA opr,X LDA opr,X LDA ,X LDA opr,SP LDA opr,SP	将存储器数值送到累加器	A←(M)	0	—	—	↕	↕	—	IMM DIR EXT IX2 IX1 IX SP1 SP2	A6 B6 C6 D6 E6 F6 9EE6 9ED6	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

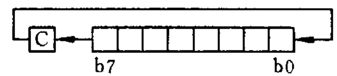
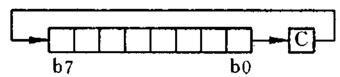
续表

指令助记符	操　作	功能描述	条件码 V	H	I	N	Z	C	寻址方式	操作码	操作数	周期数
LDHX #opr	将存储器数值送到变址寄存器	H:X←(M:M+$0001)	0	—	—	↕	↕	—	IMM	45	ii jj	3
LDHX opr									DIR	55	dd	4
LDX #opr	将存储器数值送到X寄存器	X←(M)	0	—	—	↕	↕	—	IMM	AE	ii	2
LDX opr									DIR	BE	dd	3
LDX opr									EXT	CE	hh ll	4
LDX opr,X									IX2	DE	ee ff	4
LDX opr,X									IX1	EE	ff	3
LDX ,X									IX	FE		2
LDX opr,SP									SP1	9EEE	ff	4
LDX opr,SP									SP2	9EDE	ee ff	5
LSL opr	逻辑左移		↕	—	—	↕	↕	↕	DIR	38	dd	4
LSLA									INH	48		1
LSLX									INH	58		1
LSL opr,X									IX1	68	ff	4
LSL ,X									IX	78		3
LSL opr,SP									SP1	9E68	ff	5
LSR opr	逻辑右移		↕	—	—	0	↕	↕	DIR	34	dd	4
LSRA									INH	44		1
LSRX									INH	54		1
LSR opr,X									IX1	64	ff	4
LSR ,X									IX	74		3
LSR opr,SP									SP1	9E64	ff	5
MOV opr,opr	移动指令	$(M)_{目的}$←$(M)_{源}$	0	—	—	↕	↕	—	DD	4E	dd dd	5
MOV opr,X+									DIX+	5E	dd	4
MOV #opr,opr									IMD	6E	ii dd	4
MOV X+,opr									IX+D	7E	dd	4

续表

指令助记符	操作	功能描述	条件码						寻址方式	操作码	操作数	周期数
			V	H	I	N	Z	C				
MUL	无符号数乘法指令	X:A←(X)×(A)	—	0	—	—	—	0	INH	42		5
NEG opr NEGA NEGX NEG opr,X NEG ,X NEG opr,SP	求补指令	M←−(M)=$00−(M) A←−(A)=$00−(A) X←−(X)=$00−(X) M←−(M)=$00−(M) M←−(M)=$00−(M)	↕	—	—	↕	↕	↕	DIR INH INH IX1 IX SP1	30 40 50 60 70 9E60	dd ff ff	4 1 1 4 3 5
NOP	空操作	无	—	—	—	—	—	—	INH	9D		1
NSA	累加器半字节交换	A←(A[3:0]:A[7:4])	—	—	—	—	—	—	INH	62		3
ORA #opr ORA opr ORA opr ORA opr,X ORA opr,X ORA ,X ORA opr,SP ORA opr,SP	累加器与存储器"或"	A←(A)和(M)或运算结果	0	—	—	↕	↕	—	IMM DIR EXT IX2 IX1 IX SP1 SP2	AA BA CA DA EA FA 9EEA 9EDA	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
PSHA	累加器内容进栈	推(A)入栈;SP←(SP)−$0001	—	—	—	—	—	—	INH	87		2
PSHH	H寄存器内容进栈	推(H)入栈;SP←(SP)−$0001	—	—	—	—	—	—	INH	8B		2
PSHX	X寄存器内容进栈	推(X)入栈;SP←(SP)−$0001	—	—	—	—	—	—	INH	89		2
PULA	累加器内容出栈	SP←(SP+$0001);弹出(A)	—	—	—	—	—	—	INH	86		2
PULH	H寄存器内容出栈	SP←(SP+$0001);弹出(H)	—	—	—	—	—	—	INH	8A		2
PULX	X寄存器内容出栈	SP←(SP+$0001);弹出(X)	—	—	—	—	—	—	INH	88		2

续表

指令助记符	操　作	功能描述	条件码						寻址方式	操作码	操作数	周期数
			V	H	I	N	Z	C				
ROL opr ROLA ROLX ROL opr,X ROL ,X ROL opr,SP	包括进位标志的循环左移		↕	—	—	↕	↕	↕	DIR INH INH IX1 IX SP1	39 49 59 69 79 9E69	dd ff ff	4 1 1 4 3 5
ROR opr RORA RORX ROR opr,X ROR ,X ROR opr,SP	包括进位标志的循环右移		↕	—	—	↕	↕	↕	DIR INH INH IX1 IX SP1	36 46 56 66 76 9E66	dd ff ff	4 1 1 4 3 5
RSP	堆栈指针复位	SP← $FF	—	—	—	—	—	—	INH	9C		1
RTI	中断返回	SP←(SP)+ $0001;弹出(CCR) SP←(SP)+ $0001;弹出(A) SP←(SP)+ $0001;弹出(X) SP←(SP)+ $0001;弹出(PCH) SP←(SP)+ $0001;弹出(PCL)	↕	↕	↕	↕	↕	↕	INH	80		7
RTS	子程序返回	SP←(SP)+ $0001;弹出(PCH) SP←(SP)+ $0001;弹出(PCL)	—	—	—	—	—	—	INH	81		4

续表

指令助记符	操作	功能描述	条件码 V	H	I	N	Z	C	寻址方式	操作码	操作数	周期数
SBC #opr	包括进位标志的减法指令	A←(A)−(M)−(C)	↕	—	—	↕	↕	↕	IMM	A2	ii	2
SBC opr									DIR	B2	dd	3
SBC opr									EXT	C2	hh ll	4
SBC opr,X									IX2	D2	ee ff	4
SBC opr,X									IX1	E2	ff	3
SBC ,X									IX	F2		2
SBC opr,SP									SP1	9EE2	ff	4
SBC opr,SP									SP2	9ED2	ee ff	5
SEC	进位标志置 1	C←1	—	—	—	—	—	1	INH	99		1
SEI	中断屏蔽位置 1	I←1	—	—	1	—	—	—	INH	9B		2
STA opr	累加器内容送到存储器	M←(A)	0	—	—	↕	↕	—	DIR	B7	dd	3
STA opr									EXT	C7	hh ll	4
STA opr,X									IX2	D7	ee ff	4
STA opr,X									IX1	E7	ff	3
STA ,X									IX	F7		2
STA opr,SP									SP1	9EE7	ff	4
STA opr,SP									SP2	9ED7	ee ff	5
STHX opr	变址寄存器内容送到目标地址	(M:M+$0001)←(H:X)	0	—	—	↕	↕	—	DIR	35	dd	4
STOP	允许$\overline{IRQ}$引脚中断,停止晶振	1 bit←0;停止晶振	—	—	0	—	—	—	INH	8E		1

续表

指令助记符	操　　作	功能描述	条件码 V	H	I	N	Z	C	寻址方式	操作码	操作数	周期数
STX opr	X寄存器内容送到存储器	M←(X)	0	—	—	↕	↕	—	DIR	BF	dd	3
STX opr									EXT	CF	hh ll	4
STX opr,X									IX2	DF	ee ff	4
STX opr,X									IX1	EF	ff	3
STX ,X									IX	FF		2
STX opr,SP									SP1	9EEF	ff	4
STX opr,SP									SP2	9EDF	ee ff	5
SUB #opr	减法指令	A←(A)−(M)	↕	—	—	↕	↕	↕	IMM	A0	ii	2
SUB opr									DIR	B0	dd	3
SUB opr									EXT	C0	hh ll	4
SUB opr,X									IX2	D0	ee ff	4
SUB opr,X									IX1	E0	ff	3
SUB ,X									IX	F0		2
SUB opr,SP									SP1	9EE0	ff	4
SUB opr,SP									SP2	9ED0	ee ff	5
SWI	软中断指令	PC←(PC)+＄0001;(PCL)入栈 SP←(SP)−＄0001;(PCH)入栈 SP←(SP)−＄0001;(X)入栈 SP←(SP)−＄0001;(A)入栈 SP←(SP)−＄0001;(CCR)入栈 SP←(SP)−＄0001;I←1 PCH←中断矢量高字节 PCL←中断矢量低字节	—	—	1	—	—	—	INH	83		9

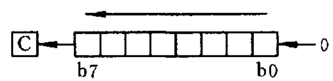
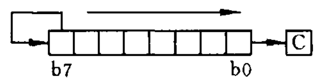
续表

指令助记符	操　　作	功 能 描 述	条 件 码 V	H	I	N	Z	C	寻址方式	操作码	操作数	周期数
TAP	累加器内容送到条件码寄存器	CCR←(A)	↕	↕	↕	↕	↕	↕	INH	84		2
TAX	累加器内容送到 X 寄存器	X←(A)	—	—	—	—	—	—	INH	97		1
TPA	条件码寄存器内容 送到累加器	A←(CCR)	—	—	—	—	—	—	INH	85		1
TST opr TSTA TSTX TST opr,X TST ,X TST opr,SP	符号及零标志的测试	(A)－＄00 (X)－＄00 (M)－＄00	0	—	—	↕	↕	—	DIR INH INH IX1 IX SP1	3D 4D 5D 6D 7D 9E6D	dd ff ff	3 1 1 3 2 4
TSX	堆栈指针送到变址寄存器	H:X←(SP)＋＄0001	—	—	—	—	—	—	INH	95		2
TXA	X 寄存器内容送到累加器	A←(X)	—	—	—	—	—	—	INH	9F		1
TXS	变址寄存器内容送到 堆栈指针	(SP)←(H:X)－＄0001	—	—	—	—	—	—	INH	94		2
WAIT	允许中断，停止处理器	I bit←0	—	—	0	—	—	—	INH	8F		1

英文说明

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
ADC #opr	Add with Carry	A←(A)+(M)+(C)	↕	↕	—	↕	↕	↕	IMM	A9	ii	2
ADC opr									DIR	B9	dd	3
ADC opr									EXT	C9	hh ll	4
ADC opr,X									IX2	D9	ee ff	4
ADC opr,X									IX1	E9	ff	3
ADC ,X									IX	F9		2
ADC opr,SP									SP1	9EE9	ff	4
ADC opr,SP									SP2	9ED9	ee ff	5
ADD #opr	Add without Carry	A←(A)+(M)	↕	↕	—	↕	↕	↕	IMM	AB	ii	2
ADD opr									DIR	BB	dd	3
ADD opr									EXT	CB	hh ll	4
ADD opr,X									IX2	DB	ee ff	4
ADD opr,X									IX1	EB	ff	3
ADD ,X									IX	FB		2
ADD opr,SP									SP1	9EEB	ff	4
ADD opr,SP									SP2	9EDB	ee ff	5
AIS #opr	Add Immediate Value (Signed) to Stack Pointer	SP←(SP)+(16«M)	—	—	—	—	—	—	IMM	A7	ii	2
AIX #opr	Add Immediate Value (Signed) to Index Register (H:X)	H:X←(H:X)+(16«M)	—	—	—	—	—	—	IMM	AF	ii	2
AND #opr	Logical AND	A←(A) & (M)	0	—	—	↕	↕	—	IMM	A4	ii	2
AND opr									DIR	B4	dd	3
AND opr									EXT	C4	hh ll	4
AND opr,X									IX2	D4	ee ff	4
AND opr,X									IX1	E4	ff	3
AND ,X									IX	F4		2
AND opr,SP									SP1	9EE4	ff	4
AND opr,SP									SP2	9ED4	ee ff	5

续表

Source Form	Operation	Description	Effect on CCR						Address Mode	Opco-de	Oper-and	Cyc-les
			V	H	I	N	Z	C				
ASL opr ASLA ASLX ASL opr,X ASL ,X ASL opr,SP	Arithmetic Shift Left (Same as LSL)		↕	—	—	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5
ASR opr ASRA ASRX ASR opr,X ASR opr,X ASR opr,SP	Arithmetic Shift Right		↕	—	—	↕	↕	↕	DIR INH INH IX1 IX SP1	37 47 57 67 77 9E67	dd ff ff	4 1 1 4 3 5
BCC rel	Branch if Carry Bit Clear	PC←(PC)+ $ 0002+rel? (C)=0	—	—	—	—	—	—	REL	24	rr	3
BCLR n,opr	Clear Bit n in Memory	Mn←0	—	—	—	—	—	—	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	11 13 15 17 19 1B 1D 1F	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BCS rel	Branch if Carry Bit Set (Same as BLO)	PC←(PC)+ $ 0002+rel? (C)=1	—	—	—	—	—	—	REL	25	rr	3
BEQ rel	Branch if Equal	PC←(PC)+ $ 0002+rel? (Z)=1	—	—	—	—	—	—	REL	27	rr	3

续表

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
BGE opr	Branch if Greater Than or Equal To (Signed Operands)	PC←(PC)+$0002+rel? (N⊕V)=0	—	—	—	—	—	—	REL	90	rr	3
BGT opr	Branch if Greater Than (Signed Operands)	PC←(PC)+$0002+rel? (Z)|(N⊕V)=0	—	—	—	—	—	—	REL	92	rr	3
BHCC rel	Branch if Half Carry Bit Clear	PC←(PC)+$0002+rel? (H)=0	—	—	—	—	—	—	REL	28	rr	3
BHCS rel	Branch if Half Carry Bit Set	PC←(PC)+$0002+rel? (H)=1	—	—	—	—	—	—	REL	29	rr	3
BHI rel	Branch if Higher	PC←(PC)+$0002+rel? (C)|(Z)=0	—	—	—	—	—	—	REL	22	rr	3
BHS rel	Branch if Higher or Same (Same as BCC)	PC←(PC)+$0002+rel? (C)=0	—	—	—	—	—	—	REL	24	rr	3
BIH rel	Branch if $\overline{IRQ}$ Pin High	PC←(PC)+$0002+rel? $\overline{IRQ}$=1	—	—	—	—	—	—	REL	2F	rr	3
BIL rel	Branch if $\overline{IRQ}$ Pin Low	PC←(PC)+$0002+rel? $\overline{IRQ}$=0	—	—	—	—	—	—	REL	2E	rr	3
BIT #opr BIT opr BIT opr BIT opr,X BIT opr,X BIT,X BIT opr,SP BIT opr,SP	Bit Test	(A) & (M)	0	—	—	↕	↕	—	IMM DIR EXT IX2 IX1 IX SP1 SP2	A5 B5 C5 D5 E5 F5 9EE5 9ED5	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
BLE opr	Branch if Less Than or Equal To (Signed Operands)	PC←(PC)+$0002+rel? (Z)|(N⊕V)=1	—	—	—	—	—	—	REL	93	rr	3
BLO rel	Branch if Lower (Same as BCS)	PC←(PC)+$0002+rel? (C)=1	—	—	—	—	—	—	REL	25	rr	3
BLS rel	Branch if Lower or Same	PC←(PC)+$0002+rel? (C)|(Z)=1	—	—	—	—	—	—	REL	23	rr	3

续表

Source Form	Operation	Description	Effect on CCR						Address Mode	Opco-de	Oper-and	Cyc-les
			V	H	I	N	Z	C				
BLT opr	Branch if Less Than (Signed Op-erands)	PC←(PC)+$0002+rel? (N⊕V)=1	—	—	—	—	—	—	REL	91	rr	3
BMC rel	Branch if Interrupt Mask Clear	PC←(PC)+$0002+rel? (I)=0	—	—	—	—	—	—	REL	2C	rr	3
BMI rel	Branch if Minus	PC←(PC)+$0002+rel? (N)=1	—	—	—	—	—	—	REL	2B	rr	3
BMS rel	Branch if Interrupt Mask Set	PC←(PC)+$0002+rel? (I)=1	—	—	—	—	—	—	REL	2D	rr	3
BNE rel	Branch if Not Equal	PC←(PC)+$0002+rel? (Z)=0	—	—	—	—	—	—	REL	26	rr	3
BPL rel	Branch if Plus	PC←(PC)+$0002+rel? (N)=0	—	—	—	—	—	—	REL	2A	rr	3
BRA rel	Branch Always	PC←(PC)+$0002+rel	—	—	—	—	—	—	REL	20	rr	3
BRCLR n, opr, rel	Branch if Bit n in Memory Clear	PC←(PC)+$0003+rel? (Mn)=0	—	—	—	—	—	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 03 05 07 09 0B 0D 0F	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BRN rel	Branch Never	PC←(PC)+$0002	—	—	—	—	—	—	REL	21	rr	3

续表

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
BRSET n, opr, rel	Branch if Bit n in Memory Set	PC←(PC)+$0003+rel? (Mn)=1	—	—	—	—	—	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 02 04 06 08 0A 0C 0E	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BSET n,opr	Set Bit n in Memory	Mn←1	—	—	—	—	—	—	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	10 12 14 16 18 1A 1C 1E	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BSR rel	Branch to Subroutine	PC←(PC)+$0002;push (PCL) SP←(SP)−$0001;push (PCH) SP←(SP)−$0001 PC←(PC)+rel	—	—	—	—	—	—	REL	AD	rr	4
CBEQ opr,rel CBEQA #opr,rel CBEQX #opr,rel CBEQ opr,X+,rel CBEQ X+,rel CBEQ opr,SP,rel	Compare and Branch if Equal	PC←(PC)+$0003+rel? (A)−(M)=$00 PC←(PC)+$0003+rel? (A)−(M)=$00 PC←(PC)+$0003+rel? (X)−(M)=$00 PC←(PC)+$0003+rel? (A)−(M)=$00 PC←(PC)+$0002+rel? (A)−(M)=$00 PC←(PC)+$0004+rel? (A)−(M)=$00	—	—	—	—	—	—	DIR IMM IMM IX1+ IX+ SP1	31 41 51 61 71 9E61	dd rr ii rr ii rr ff rr rr ff rr	5 4 4 5 4 6

续表

Source Form	Operation	Description	Effect on CCR						Address Mode	Opco-de	Oper-and	Cyc-les
			V	H	I	N	Z	C				
CLC	Clear Carry Bit	C←0	—	—	—	—	—	0	INH	98		1
CLI	Clear Interrupt Mask Bit	I←0	—	—	0	—	—	—	INH	9A		2
CLR opr	Clear	M←$00	0	—	—	0	1	—	DIR	3F	dd	3
CLRA		A←$00							INH	4F		1
CLRX		X←$00							INH	5F		1
CLRH		H←$00							INH	8C		1
CLR opr,X		M←$00							IX1	6F	ff	3
CLR ,X		M←$00							IX	7F		2
CLR opr,SP		M←$00							SP1	9E6F	ff	4
CMP #opr	Compare Accumulator with Memory	(A)−(M)	↕	—	—	↕	↕	↕	IMM	A1	ii	2
CMP opr									DIR	B1	dd	3
CMP opr									EXT	C1	hh ll	4
CMP opr,X									IX2	D1	ee ff	4
CMP opr,X									IX1	E1	ff	3
CMP ,X									IX	F1		2
CMP opr,SP									SP1	9EE1	ff	4
CMP opr,SP									SP2	9ED1	ee ff	5
COM opr	Complement (One's Complement)	M←($\overline{M}$)=$FF−(M)	0	—	—	↕	↕	1	DIR	33	dd	4
COMA		A←($\overline{A}$)=$FF−(M)							INH	43		1
COMX		X←($\overline{X}$)=$FF−(M)							INH	53		1
COM opr,X		M←($\overline{M}$)=$FF−(M)							IX1	63	ff	4
COM ,X		M←($\overline{M}$)=$FF−(M)							IX	73		3
COM opr,SP		M←($\overline{M}$)=$FF−(M)							SP1	9E63	ff	5

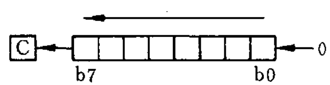
续表

Source Form	Operation	Description	Effect on CCR V	H	I	N	Z	C	Address Mode	Opcode	Operand	Cycles
CPHX #opr CPHX opr	Compare Index Register (H:X) with Memory	(H:X)−(M:M+ $0001)	↕	—	—	↕	↕	↕	IMM DIR	65 75	ii ii+1 dd	3 4
CPX #opr CPX opr CPX opr CPX ,X CPX opr,X CPX opr,X CPX opr,SP CPX opr,SP	Compare X (Index Register Low) with Memory	(X)−(M)	↕	—	—	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A3 B3 C3 D3 E3 F3 9EE3 9ED3	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
DAA	Decimal Adjust Accumulator	$(A)_{10}$	U	—	—	↕	↕	↕	INH	72		2
DBNZ opr,rel DBNZA rel DBNZX rel DBNZ opr,X,rel DBNZ X,rel DBNZ opr,SP,rel	Decrement and Branch if Not Zero	A←(A)− $0001 or M←(M)− $01 or X←(X)− $0001 PC←(PC)+ $0003+rel if (result)≠0 for DBNZ direct,IX1 PC←(PC)+ $0002+rel if (result)≠0 for DBNZA,DBNZX,or IX PC←(PC)+ $0004+rel if (result)≠0 for DBNZ SP1	—	—	—	—	—	—	DIR INH INH IX1 IX SP1	3B 4B 5B 6B 7B 9E6B	dd rr rr rr ff rr rr ff rr	5 3 3 5 4 6
DEC opr DECA DECX DEC opr,X DEC ,X DEC opr,SP	Decrement	M←(M)− $01 A←(A)− $01 X←(X)− $01 M←(M)− $01 M←(M)− $01 M←(M)− $01	↕	—	—	↕	↕	—	DIR INH INH IX1 IX SP1	3A 4A 5A 6A 7A 9E6A	dd ff ff	4 1 1 4 3 5
DIV	Divide	A←(H:A)/(X) H←Remainder	—	—	—	—	↕	↕	INH	52		7

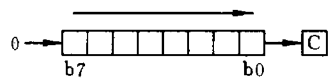
续表

Source Form	Operation	Description	Effect on CCR V	H	I	N	Z	C	Address Mode	Opcode	Operand	Cycles
EOR #opr	Exclusive OR Memory with Accumulator	A←(A⊕M)	0	—	—	↕	↕	—	IMM	A8	ii	2
EOR opr									DIR	B8	dd	3
EOR opr									EXT	C8	hh ll	4
EOR opr,X									IX2	D8	ee ff	4
EOR opr,X									IX1	E8	ff	3
EOR ,X									IX	F8		2
EOR opr,SP									SP1	9EE8	ff	4
EOR opr,SP									SP2	9ED8	ee ff	5
INC opr	Increment	M←(M)+$01	↕	—	—	↕	↕	—	DIR	3C	dd	4
INCA		A←(A)+$01							INH	4C		1
INCX		X←(X)+$01							INH	5C		1
INC opr,X		M←(M)+$01							IX1	6C	ff	4
INC ,X		M←(M)+$01							IX	7C		3
INC opr,SP		M←(M)+$01							SP1	9E6C	ff	5
JMP opr	Jump	PC←Jump Address	—	—	—	—	—	—	DIR	BC	dd	2
JMP opr									EXT	CC	hh ll	3
JMP opr,X									IX2	DC	ee ff	4
JMP opr,X									IX1	EC	ff	3
JMP ,X									IX	FC		2
JSR opr	Jump to Subroutine	PC←(PC)+n(n=1,2,or 3) Push (PCL);SP←(SP)−$0001 Push (PCH);SP←(SP)−$0001 PC←Unconditional Address	—	—	—	—	—	—	DIR	BD	dd	4
JSR opr									EXT	CD	hh ll	5
JSR opr,X									IX2	DD	ee ff	6
JSR opr,X									IX1	ED	ff	5
JSR ,X									IX	FD		4

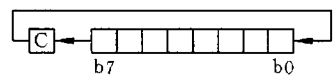
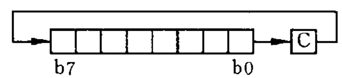
续表

Source Form	Operation	Description	Effect on CCR V	H	I	N	Z	C	Address Mode	Opcode	Operand	Cycles
LDA #opr	Load Accumulator from Memory	A←(M)	0	—	—	↕	↕	—	IMM	A6	ii	2
LDA opr									DIR	B6	dd	3
LDA opr									EXT	C6	hh ll	4
LDA opr,X									IX2	D6	ee ff	4
LDA opr,X									IX1	E6	ff	3
LDA ,X									IX	F6		2
LDA opr,SP									SP1	9EE6	ff	4
LDA opr,SP									SP2	9ED6	ee ff	5
LDHX #opr	Load Index Register (H:X) from Memory	H:X←(M:M+$0001)	0	—	—	↕	↕	—	IMM	45	ii jj	3
LDHX opr									DIR	55	dd	4
LDX #opr	Load X (Index Register Low) from Memory	X←(M)	0	—	—	↕	↕	—	IMM	AE	ii	2
LDX opr									DIR	BE	dd	3
LDX opr									EXT	CE	hh ll	4
LDX opr,X									IX2	DE	ee ff	4
LDX opr,X									IX1	EE	ff	3
LDX ,X									IX	FE		2
LDX opr,SP									SP1	9EEE	ff	4
LDX opr,SP									SP2	9EDE	ee ff	5
LSL opr	Logical Shift Left (Same as ASL)		↕	—	—	↕	↕	↕	DIR	38	dd	4
LSLA									INH	48		1
LSLX									INH	58		1
LSL opr,X									IX1	68	ff	4
LSL ,X									IX	78		3
LSL opr,SP									SP1	9E68	ff	5

续表

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
LSR opr LSRA LSRX LSR opr,X LSR ,X LSR opr,SP	Logical Shift Right		↕	—	—	0	↕	↕	DIR INH INH IX1 IX SP1	34 44 54 64 74 9E64	dd ff ff	4 1 1 4 3 5
MOV opr,opr MOV opr,X+ MOV #opr,opr MOV X+,opr	Move	$(M)_{destination} \leftarrow (M)_{source}$ H:X←(H:X)+ $001 in IX+D and DIX+Modes	0	—	—	↕	↕	—	DD DIX+ IMD IX+D	4E 5E 6E 7E	dd dd dd ii dd dd	5 4 4 4
MUL	Unsigned multiply	X:A←(X)×(A)	—	0	—	—	—	0	INH	42		5
NEG opr NEGA NEGX NEG opr,X NEG ,X NEG opr,SP	Negate (Two's Complement)	M←−(M)= $00−(M) A←−(A)= $00−(A) X←−(X)= $00−(X) M←−(M)= $00−(M) M←−(M)= $00−(M)	↕	—	—	↕	↕	↕	DIR INH INH IX1 IX SP1	30 40 50 60 70 9E60	dd ff ff	4 1 1 4 3 5
NOP	No Operation	None	—	—	—	—	—	—	INH	9D		1
NSA	Nibble Swap Accumulator	A←(A[3:0]:A[7:4])	—	—	—	—	—	—	INH	62		3
ORA #opr ORA opr ORA opr ORA opr,X ORA opr,X ORA ,X ORA opr,SP ORA opr,SP	Inclusive OR Accumulator and Memory	A←(A)|(M)	0	—	—	↕	↕	—	IMM DIR EXT IX2 IX1 IX SP1 SP2	AA BA CA DA EA FA 9EEA 9EDA	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

续表

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
PSHA	Push Accumulator onto Stack	Push (A); SP←(SP) − $0001	—	—	—	—	—	—	INH	87		2
PSHH	Push H (Index Register High) onto Stack	Push (H); SP←(SP) − $0001	—	—	—	—	—	—	INH	8B		2
PSHX	Push X (Index Register Low) onto Stack	Push (X); SP←(SP) − $0001	—	—	—	—	—	—	INH	89		2
PULA	Pull Accumulator from Stack	SP←(SP + $0001); Pull (A)	—	—	—	—	—	—	INH	86		2
PULH	Pull H (Index Register High) from Stack	SP←(SP + $0001); Pull (H)	—	—	—	—	—	—	INH	8A		2
PULX	Pull X (Index Register Low) from Stack	SP←(SP + $0001); Pull (X)	—	—	—	—	—	—	INH	88		2
ROL opr ROLA ROLX ROL opr,X ROL ,X ROL opr,SP	Rotate Left through Carry		↕	—	—	↕	↕	↕	DIR INH INH IX1 IX SP1	39 49 59 69 79 9E69	dd ff ff	4 1 1 4 3 5
ROR opr RORA RORX ROR opr,X ROR ,X ROR opr,SP	Rotate Right through Carry		↕	—	—	↕	↕	↕	DIR INH INH IX1 IX SP1	36 46 56 66 76 9E66	dd ff ff	4 1 1 4 3 5
RSP	Reset Stack Pointer	SP← $FF	—	—	—	—	—	—	INH	9C		1

续表

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
RTI	Return from Interrupt	SP←(SP)+$0001;Pull (CCR) SP←(SP)+$0001;Pull (A) SP←(SP)+$0001;Pull (X) SP←(SP)+$0001;Pull (PCH) SP←(SP)+$0001;Pull (PCL)	↕	↕	↕	↕	↕	↕	INH	80		7
RTS	Return from Subroutine	SP←SP+$0001;Pull (PCH) SP←SP+$0001;Pull (PCL)	—	—	—	—	—	—	INH	81		4
SBC #opr	Subtract with Carry	A←(A)−(M)−(C)	↕	—	—	↕	↕	↕	IMM	A2	ii	2
SBC opr									DIR	B2	dd	3
SBC opr									EXT	C2	hh ll	4
SBC opr,X									IX2	D2	ee ff	4
SBC opr,X									IX1	E2	ff	3
SBC ,X									IX	F2		2
SBC opr,SP									SP1	9EE2	ff	4
SBC opr,SP									SP2	9ED2	ee ff	5
SEC	Set Carry Bit	C←1	—	—	—	—	—	1	INH	99		1
SEI	Set Interrupt Mask Bit	I←1	—	—	1	—	—	—	INH	9B		2
STA opr	Store Accumulator in Memory	M←(A)	0	—	—	↕	↕	—	DIR	B7	dd	3
STA opr									EXT	C7	hh ll	4
STA opr,X									IX2	D7	ee ff	4
STA opr,X									IX1	E7	ff	3
STA ,X									IX	F7		2
STA opr,SP									SP1	9EE7	ff	4
STA opr,SP									SP2	9ED7	ee ff	5

续表

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
STHX opr	Store H:X (Index Reg.)	(M:M+ $0001)←(H:X)	0	—	—	↕	↕	—	DIR	35	dd	4
STOP	Enable $\overline{IRQ}$ pin; Stop Osc.	I bit←0; Stop Oscillator	—	—	0	—	—	—	INH	8E		1
STX opr STX opr STX opr,X STX opr,X STX ,X STX opr,SP STX opr,SP	Store X (Index Register Low) in Memory	M←(X)	0	—	—	↕	↕	—	DIR EXT IX2 IX1 IX SP1 SP2	BF CF DF EF FF 9EEF 9EDF	dd hh ll ee ff ff ff ee ff	3 4 4 3 2 4 5
SUB #opr SUB opr SUB opr SUB opr,X SUB opr,X SUB ,X SUB opr,SP SUB opr,SP	Subtract	A←(A)−(M)	↕	—	—	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A0 B0 C0 D0 E0 F0 9EE0 9ED0	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
SWI	Software Interrupt	PC←(PC)+ $0001; Push (PCL) SP←(SP)− $0001; Push (PCH) SP←(SP)− $0001; Push (X) SP←(SP)− $0001; Push (A) SP←(SP)− $0001; Push (CCR) SP←(SP)− $0001; I←1 PCH←Interrupt Vector High Byte PCL←Interrupt Vector Low Byte	—	—	1	—	—	—	INH	83		9

续表

Source Form	Operation	Description	Effect on CCR						Address Mode	Opco-de	Oper-and	Cyc-les
			V	H	I	N	Z	C				
TAP	Transfer Accumulator to CCR	CCR←(A)	↕	↕	↕	↕	↕	↕	INH	84		2
TAX	Transfer Accumulator to X (Index Register Low)	X←(A)	—	—	—	—	—	—	INH	97		1
TPA	Transfer CCR to Accumulator	A←(CCR)	—	—	—	—	—	—	INH	85		1
TST opr TSTA TSTX TST opr,X TST ,X TST opr,SP	Test for Negative or Zero	(A)－$00 (X)－$00 (M)－$00	0	—	—	↕	↕	—	DIR INH INH IX1 IX SP1	3D 4D 5D 6D 7D 9E6D	dd ff ff	3 1 1 3 2 4
TSX	Transfer SP to Index Reg.	H:X←(SP)＋$0001	—	—	—	—	—	—	INH	95		2
TXA	Transfer X(Index Reg. Low) to Accumulator	A←(X)	—	—	—	—	—	—	INH	9F		1
TXS	Transfer Index Reg. to SP	(SP)←(H:X)－$0001	—	—	—	—	—	—	INH	94		2
WAIT	Enable Interrupts;Stop Processor	I bit←0	—	—	0	—	—	—	INH	8F		1

7. 参数极值表

特性	符号	取值	单位
电源电压	V_{DD}	−0.3 至 +6.0	V
输入电压	V_{In}	V_{SS}−0.3 至 V_{DD}+0.3	V
除 V_{DD}、V_{SS} 及 PTC0～PTC4 外，每个管脚的最大电流	I	±15	mA
PTC0～PTC4 管脚的最大电流	I_{PTC0}～I_{PTC4}	±25	mA
V_{DD} 管脚的最大电流	I_{mvdd}	150	mA
V_{SS} 管脚的最大电流	I_{mvss}	150	mA
贮存温度	T_{stg}	−55 至 +150	℃
工作温度范围	T_A	−40 至 +85	℃
工作电压范围	V_{DD}	3.0±10% 5.0±10%	V

注：电压值为相对于 V_{SS} 的值。

8. 直流电气特性

5.0V 电气特性

特　　性[1]	符号	最小值	典型值[2]	最大值	单位
输出高电平					
(I_{Load}=−2.0mA) 所有 I/O 管脚	V_{OH}	$V_{DD}-0.8$	—	—	V
(I_{Load}=−10.0mA) 所有 I/O 管脚	V_{OH}	$V_{DD}-1.5$	—	—	V
(I_{Load}=−10.0mA) 仅 PTC0～PTC4	V_{OH}	$V_{DD}-0.8$	—	—	V
PORTC,PORTE,PTD0～PTD3 的最大总电流	I_{OH1}	—	—	50	mA
PORTA,PORTB,PTD4～PTD7 的最大总电流	I_{OH2}	—	—	50	mA
所有端口管脚的总电流	I_{OHT}	—	—	100	mA
输出低电平					
(I_{Load}=1.6mA) 所有 I/O 管脚	V_{OL}	—	—	0.4	V
(I_{Load}=10mA) 所有 I/O 管脚	V_{OL}	—	—	1.5	V
(I_{Load}=15mA) 仅 PTC0～PTC4	V_{OL}	—	—	1.0	V
PORTC,PORTE,PTD0～PTD3 的最大组合电流	I_{OL1}	—	—	50	mA
PORTA,PORTB,PTD4～PTD7 的最大组合电流	I_{OL2}	—	—	50	mA
所有端口管脚的总电流	I_{OLT}	—	—	100	mA
输入高电平 所有端口、IRQ、RESET、OSC1 管脚	V_{IH}	$0.7\times V_{DD}$	—	V_{DD}	V
输入低电平 所有端口、IRQ、RESET、OSC1 管脚	V_{IL}	V_{SS}	—	$0.2\times V_{DD}$	V
V_{DD}电源电流	I_{DD}				
运行态 (RUN)[3]		—	—	30	mA
等待态 (WAIT)[4]		—	—	12	mA
停止态 (STOP)[5]		—	5	—	μA
25℃					
25℃(允许定时基模块 TBM)[6]		—	20	—	μA
25℃(允许低电压禁止模块 LVI 及定时基模块 TBM)[6]		—	300	—	μA
−40℃～85℃(允许定时基模块 TBM)[6]		—	50	—	μA
−40℃～85℃(允许低电压禁止模块 LVI 及定时基模块 TBM)[6]		—	500	—	μA
I/O 端口高阻漏电流[7]	I_{IL}	—	—	±10	μA
输入电流	I_{In}	—	—	±1	μA

续表

特　　性[1]	符号	最小值	典型值[2]	最大值	单位
上拉电阻(作输入时) 端口 PTA7/KBD7～PTA0/KBD0,PTC6～PTC0 PTD7/T2CH1～PTD0/$\overline{SS}$	R_{PU}	20	33	50	kΩ
电容 端口(输入或输出时)	C_{Out} C_{In}	— —	— —	12 8	pF
监控模式输入电平	V_{TST}	$V_{DD}+2.5$	—	9	V

3.0V 电气特性

特　　性[1]	符号	最小值	典型值[2]	最大值	单位
输出高电平					
($I_{Load}=-0.6mA$)所有 I/O 管脚	V_{OH}	$V_{DD}-0.3$	—	—	V
($I_{Load}=-4.0mA$)所有 I/O 管脚	V_{OH}	$V_{DD}-1.0$	—	—	V
($I_{Load}=-4.0mA$)仅 PTC0～PTC4	V_{OH}	$V_{DD}-0.5$	—	—	V
PORTC,PORTE,PTD0～PTD3 的最大总电流	I_{OH1}	—	—	30	mA
PORTA,PORTB,PTD4～PTD7 的最大总电流	I_{OH2}	—	—	30	mA
所有端口管脚的总电流	I_{OHT}	—	—	60	mA
输出低电平					
($I_{Load}=0.5mA$)所有 I/O 管脚	V_{OL}	—	—	0.3	V
($I_{Load}=6.0mA$)所有 I/O 管脚	V_{OL}	—	—	1.0	V
($I_{Load}=10.0mA$)仅 PTC0～PTC4	V_{OL}	—	—	0.8	V
PORTC,PORTE,PTD0～PTD3 的最大组合电流	I_{OL1}	—	—	30	mA
PORTA,PORTB,PTD4～PTD7 的最大组合电流	I_{OL2}	—	—	30	mA
所有端口管脚的总电流	I_{OLT}	—	—	60	mA
输入高电平 所有端口、IRQ、RESET、OSC1 管脚	V_{IH}	$0.7\times V_{DD}$	—	V_{DD}	V
输入低电平 所有端口、IRQ、RESET、OSC1 管脚	V_{IL}	V_{SS}	—	$0.3\times V_{DD}$	V
V_{DD}电源电流	I_{DD}				
运行态 (RUN)[3]		—	—	10	mA
等待态 (WAIT)[4]		—	—	6	mA
停止态 (STOP)[5]		—	3	—	μA
25℃					
25℃(允许定时基模块 TBM)[6]		—	12	—	μA
25℃(允许低电压禁止模块 LVI 及定时基模块 TBM)[6]		—	200	—	μA

续表

特　　性[1]	符号	最小值	典型值[2]	最大值	单位
−40℃～85℃(允许定时基模块 TSB)[6]		—	30	—	μA
−40℃～85℃(允许低电压禁止模块 LVI 及定时基模块 TBM)[6]		—	300	—	μA
I/O 端口高阻漏电流[7]	I_{IL}	—	—	±10	μA
输入电流	I_{In}	—	—	±1	μA
上拉电阻(作输入时) 端口 PTA7/KBD7～PTA0/KBD0,PTC6～PTC0 PTD7/T2CH1～PTD0/$\overline{SS}$	R_{PU}	20	33	50	kΩ
电容 端口(输入或输出时)	C_{Out} C_{In}	— —	— —	12 8	pF
监控模式输入电平	V_{TST}	$V_{DD}+2.5$	—	9	V

注:(1) 除特殊说明外,$V_{SS}=0V(dc)$,$T_A=T_L\sim T_H$。

(2) 典型值表示的是在 25℃时电压范围的中点处的平均测量值。

(3) RUN 状态下电流 I_{DD}的测量条件是:使用外部方波时钟源($f_{OSC}=16.4MHz$),无直流负载,所有输出管脚上的电容小于 100pF,OSC2 管脚上的电容 $C_1=20pF$。所有端口设置为输入方式,RUN 状态下的电流 I_{DD}与 OSC2 上的电容成线性关系,测量时所有模块都处于允许态。

(4) WAIT 状态下的 I_{DD}的测量条件是:使用外部方波时钟源,($f_{OSC}=16.4MHz$),无直流负载,所有输出管脚上的电容小于 100pF,OSC2 管脚上的电容 $C_1=20pF$。所有端口设置为输入方式,WAIT 状态下电流 I_{DD}与 OSC2 上的电容成线性关系,测量时锁相环和低电压禁止模块处于允许态。

(5) STOP 状态下的电流 I_{DD}是管脚 OSC1 在 V_{SS}状态下测量的。

(6) 在定时基模块有效情况下测量 STOP 状态下电流 I_{DD}的条件是:使用外部方波时钟源($f_{OSC}=32.8MHz$),无直流负载,所有输出管脚上的电容小于 100pF,所有的输入端设置为输入方式。

(7) 禁止上拉和下拉电阻。

9. 控制时序表

3.0V 控制时序

特　　性[1]	符号	最小值	最大值	单位
工作频率[2]				
晶振	f_{OSC}	32	100	kHz
外部时钟[3]		dc[4]	16.4	MHz
内部操作频率	f_{OP}	—	4.1	MHz
内部时钟周期(1/f_{OP})	t_{CYC}	244	—	ns
$\overline{RESET}$输入脉冲宽度低[5]	t_{IRL}	125	—	ns
$\overline{IRQ}$中断脉冲宽度低[6](沿触发)	t_{ILIH}	125	—	ns
$\overline{IRQ}$中断脉冲周期	t_{ILIL}	TBD[8]	—	t_{CYC}
16 位定时器[7]				
输入捕捉脉冲宽度	t_{TH},t_{TL}	TBD[8]	—	ns
输入捕捉周期	t_{TLTL}			t_{CYC}

5.0V 控制时序

特　　性[1]	符号	最小值	最大值	单位
工作频率[2]				
晶振	f_{OSC}	32	100	kHz
外部时钟[3]		dc[4]	32.8	MHz
内部操作频率	f_{OP}	—	8.2	MHz
内部时钟周期(1/f_{OP})	t_{CYC}	122	—	ns
$\overline{RESET}$输入脉冲宽度低[5]	t_{IRL}	50	—	ns
$\overline{IRQ}$中断脉冲宽度低[6](沿触发)	t_{ILIH}	50	—	ns
$\overline{IRQ}$中断脉冲周期	t_{ILIL}	TBD[8]	—	t_{CYC}
16 位定时器[7]				
输入捕捉脉冲宽度	t_{TH},t_{TL}	TBD[8]	—	ns
输入捕捉周期	t_{TLTL}		t_{CYC}	

注:(1) $V_{SS}=0V$ (dc);除特殊说明外,时序是用相对于 20%V_{DD}和 70%V_{SS}来表示。

(2) 详细资料参见时钟发生模块的特性。

(3) 占空比误差不大于方波的 10%。

(4) 为了正确操作,一些模块可能需要一个比直流稍大一些的很小频率的外部时钟。

(5) 为了能够识别,必须保证 RESET 信号的最小脉宽。尤其是当能引起一次复位的脉宽较窄时,这一点是很重要的。

(6) 最小脉宽是为了保证引起中断,尤其当脉宽较小时,这一点尤为重要。

(7) 最小脉宽用于保证引起中断,尤其当脉宽较小时,这一点尤为重要。

(8) 最小周期 t_{ILIL}或 t_{TLTL}不小于执行中断服务程序的周期数与 t_{CYC}的积。

68HC08 Programmers

Device	Packages Supported	Programming Adapter	Programmer Board
68HC908AB32	64 QFP	NA	M68ICS08AB Serial Programmer/In-Circuit Simulator
68HC908AB32	64 QFP	M68UPA08AB32FU64	M68HC705UPGMR or M68HC705UGANG Parallel programmer
68HC908AZ60 68HC908AS60	64 QFP-FU 52 PLCC-FN	M68PA08AT60FNFU	M68SPGMR08
68HC908BD48	44 QFB-FB	M68UPA08BD48FB44	M68HC705UPGMR or M68HC705UGANG Parallel Programmer
	42 SDIP-B	M68UPA08BD48B42	
	28 DIP-P	M68UPA08BD48P28	
68HC908GP32	40 DIP-P 44 QFP-FB	NA	M68ICS08GP Serial Programmer/In-Circuit Simulator: recommended for DIP and QFP programming
68HC908GP32	40 DIP-P	M68UPA08GP32P40	M68HC705UPGMR or M68HC705UGANG Parallel Programmer
	42 SDIP-B	M68UPA08GP32B42	
	44 QFP-FB	M68UPA08GP32FB44	
68HC908GR8 68HC908GR4	28 DIP-P 32 QFP-FA	NA	M68ICS08GR Serial Programmer/In-Circuit Simulator
68HC908JB8	20 DIP-P 20 SOIC-DW 28 SOIC-DW 42 SDIP-B 44 QFP-FB	NA	M68ICS08JB Serial Programmer/In-Circuit Simulator
68HC908JB8	20 DIP-P	M68UPA08JB8P20	M68HC705UPGMR or M68HC705UGANG Parallel Programmer
	20 SOIC-DW	M68UPA08JB8DW20	
	28 SOIC-DW	M68UPA08JB8DW28	
	42 SDIP-B	M68UPA08JB8B42	
	44 QFP-FB	M68UPA08JB8FB44	
68HC908JK3/JK1	20 DIP-P 20 SOIC-DW	NA	M68ICS08JLJK Serial Programmer/In-Circuit Simulator
68HC908JK3/JK1	20 DIP-P	M68UPA08JK3P20	M68HC705UPGMR or M68HC705UGANG Parallel programmer
	20 SOIC-DW	M68UPA08JK3DW20	
68HC908JL3	28 DIP-P 28 SOIC-DW	NA	M68ICS08JLJK Serial Programmer/In-Circuit Simulator
68HC908JL3	28 DIP-P	M68UPA08JL3P28	M68HC705UPGMR or M68HC705UGANG Parallel Programmer
	28 SOIC-DW	M68UPA08JL3DW28	
68HC908KX8 68HC908KX2	16 DIP-P 16 SOIC-DW	NA	M68ICS08KX
68HC908LD64	64 QFP-FU	M68UPA08LD64FU64	M68HC705UPGMR or M68HC705UGANG Parallel Programmer
		M68PA08LD64FU64	M68SPGMR08 Serial Programmer with NO In-Circuit Simulator
		NA	M68PGMR08DDC FLASH Programming through DDC Interface
68HC908MR32 68HC908MR16 68HC908MR8	56 SDIP-B 64 QFP-FU 32 QFP-FA	NA	M68ICS08MR Serial Programmer/In-Circuit Simulator
68HC908RK2	20 SOIC-DW	NA	M68ICS08RK2 Serial Programmer/In-Circuit Simulator
68HC908SR12	42 SDIP-B 48 QFP-FA	NA	M68ICS08SR Serial Programmer/In-Circuit Simulator
68HC908SR12	42 SDIP-B	M68UPA08SR12B42	M68HC705UPGMR or M68HC705UGANG Parallel programmer
	48 QFP-FA	M68UPA08SR12FA48	

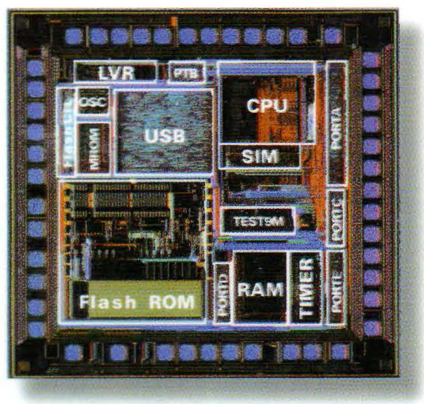
参考文献

1. 邵贝贝、刘慧银等. 微控制器原理与开发技术. 清华大学出版社，1997
2. 邵贝贝、程建平. MC68HC11 单片机开发技术. 清华大学出版社，1995
3. 第 3 届 Motorola 杯单片机设计应用大奖赛论文集.《电子产品世界》杂志出版社，2000
4. MOTOROLA. Databook
5. MOTOROLA. Application Note
6. MOTOROLA. Presentation material
7. MOTOROLA. Reference Manuals
8. MOTOROLA. Trinning material
9. Labrosse Jean J. μC/OS The Real Time Kernel. R&D，1992
10. Labrosse Jean J. μC/OS-Ⅱ The Real Time Kernel. R&D，1998

果粉藏書

刘慧银　程建平　龚光华　王云飞　许庆丰 编著

Motorola微控制器 MC68HC08 原理及其嵌入式应用



ISBN 7-302-04678-6
9 787302 046783 >
定价：26.00元